File No. UIUCDCS-F-85-945

INDUCTIVE LEARNING OF DECISION RULES WITH EXCEPTIONS: METHODOLOGY AND EXPERIMENTATION

BY

JEFFREY MARTIN BECKER

B.S., University of Illinois, 1983

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1985

Urbana, Illinois

ISG 85-14

August 1985

This research was supported in part by the National Science Foundation under grant DCR 84-06801, and in part by the Office of Naval Research under grant N00014-82-K-0186.

ACKNOWLEDGEMENTS

I would first like to thank my thesis advisor, Professor R. S. Michalski for contributing many useful ideas, comments, and support. I am grateful to Professors R. S. Michalski and P. H. Winston for allowing me to read a draft of their paper on censored production rules. Many of the ideas from their paper are used in this thesis. Thanks also go to Professor A. B. Baskin, Professor S. R. Ray, and Professor T. Brown for supplying the experimental data used for testing the system described in this thesis. Professor L. Rendell provided constructive criticism and useful information. Many members of the Intelligent Systems Group contributed suggestions, code, test data, editorial criticisms, and encouragement. Thanks go to Igor Mosetic, Tom Channic, Bob Stepp, Tony Nowicki, and Brain Falkenhainer. Thanks also to Peter Haddawy for exorcising my Lisp code.

I am grateful for the excellent facilities provided by the University of Illinois Department of Computer Science, and the Intelligent System Group. Thanks go to Tony Nowicki and Bob Stepp for keeping the ISG Sun workstations up and running.

I am especially thankful to my wife, Christine, for being understanding during the many months of late night work involved in this project, and for financial support.

This research was supported in part by the National Science Foundation under grant DCR 84-06801, and in part by the Office of Naval Research under grant N000 14-82-K-0186.

TABLE OF CONTENTS

1. INTRODUCTION 1
   1.1 Background 1
   1.2 Overview 2
   1.3 Synopsis 5

2. DESCRIPTION OF THE METHODOLOGY 6
   2.1 Background Rules 7
   2.2 Conflict Handling 8
   2.3 Learning Rules with Exceptions 10
      2.3.1 Characteristics of Rules with Unless Conditions 11
      2.3.2 Assigning a Confidence Level 13
      2.3.3 A Learning Technique 14
      2.3.4 Incremental Learning 18
   2.4 Interpreting Rules with Unless Conditions 19
   2.5 Performance Considerations in Learning 20

3. DESCRIPTION OF THE IMPLEMENTATION 24
   3.1 Conflict Handling 25
   3.2 Learning Rules with Exceptions 26
   3.3 Comparison to A^q 30
   3.4 The Rule Interpreter 31
   3.5 Performance Considerations 32

4. EXPERIMENTATION AND ANALYSIS 34
   4.1 A Description of the Applications 34
   4.2 Definitions of Measures Used 38
   4.3 Performance Comparisons 39
   4.4 The Effects of Noisy Data 43
      4.4.1 Noise in Testing Events Only 44
      4.4.2 Noise in Both Training and Testing Examples 46
      4.4.3 Noise and Approximate Decision Rules 48
      4.4.4 Discussion of Quinlan's Findings 52
   4.5 A Closed Loop Learning System 53

5. CONCLUSIONS 62
   5.1 System Performance 62
   5.2 Limitations and Future Directions 63

APPENDIX A: GLOSSARY 67

APPENDIX B: A USER'S GUIDE TO EXCEL 70

APPENDIX C: EXPERIMENTAL DATA 90

REFERENCES 120

1. INTRODUCTION

One of the goals of machine learning research is to give machines the ability to acquire useful knowledge in ways that people do. Instead of hand crafting the detailed rules needed for a knowledge based system to perform with a high level of expertise, we would like to have a system which can develop these rules from examples. This is desirable because often experts are unable to articulate the rules they use in making decisions, and in some areas there are no experts.

In the real world of measurements and decisions, very little can be known with 100% confidence. Often, it is not possible to gather all relevant information before we must make a decision. The information we gather is likely to contain errors and may contain inconsistencies. We work in a resource limited environment, exhibiting a form of *satisficing* behavior [Simon, 1960]. That is, we often stop when we find a solution which is "good enough" even though the solution is not optimal. When we make a decision, the rules we use tend to work well for the most frequently encountered problems, but unusual circumstances may require further consideration. For example, in disease diagnosis it is usual to check for a common disease associated with certain symptoms before doing expensive tests for rare diseases with similar symptoms. This thesis addresses the problems of learning approximate decision rules from imperfect data and applying these rules in a resource limited environment.

1.1. Background

This work has evolved from work on the A^q algorithm [Michalski, 1969; Michalski, 1977], and recent extensions [Michalski and Larson, 1983; Becker, 1983]. The A^q algorithm is a quasi–optimal solution to the general covering problem, originally developed for and applied to logic circuit minimisation by Michalski [1969]. It has been used for automatic acquisition of decision rules for expert systems, and conceptual data analysis. The current work extends these efforts in the directions of greater flexibility, better rule quality, and greater efficiency.

The problem of learning from noisy data has been investigated by Quinlan using a version of the ID3 algorithm modified to allow the generation of approximate decision trees [Quinlan, 1983b]. ID3 is a des-

cendent of the CLS inductive learning system [Hunt, Marin, and Stone, 1966]. Quinlan reports a numb of findings which are in part replicated in this thesis, with some interesting differences.

Another approach to inductive learning of approximate (also called *probabilistic*) decision rules described in [Rendell, 1983]. Rendell's Penetrance Learning System (PLS) is closely related to CLS a ID3. PLS produces weighted rules which can be used to determine how likely a particular event is to me a particular condition.

The idea of the *unless* condition as a useful extension to production rules was originally introduce by Winston [Winston, 1983], and elaborated by both Winston and Michalski [Michalski and Winsto 1985]. Winston's original implementation of these concepts was in a system for understanding and lear ing from stories. The current work embodies the unless condition concept in the area of learning discrim inant descriptions from examples, and in the application of rules with unless conditions under constrain on decision certainty and applied effort. Winston worked with a semantic network representation; th implementation described here uses the variable–valued logic system VL_1 [Michalski, 1974].

1.2. Overview

The task of interest here is learning of concept descriptions from examples [Michalski, 1983]. In thi paradigm, a set of training examples which have been assigned to decision classes by an expert are used a the basis for automatically inducing a general description for each decision class. The rules learned ma then be used to assign classifications to testing examples (examples for which the correct classification i unknown).

An example may be a physical object, a situation, a cause, a concept, or nearly anything else tha can be described in terms of a set of attribute–value pairs. Some learning tasks which fit this paradigr are:

(1) Establishing sense/concept associations. Given values for sensory inputs for a number of differen concepts, a rule can be learned for mapping some range of input values to each concept.

(2) Learning rules for assigning physical objects to classes given examples of physical objects from each of a finite number of classes. For example, given examples of animals from different categories in a classification hierarchy, we can learn simple rules for assigning new examples to categories.

(3) Learning condition-action rules. Given examples of when an action should and should not be applied, a generalised expression for the condition may be learned.

(4) Learning rules for fault or disease diagnosis from examples of the faults or diseases. Machine learning has already proven to be an effective means for generating knowledge bases for expert systems in this area [Michalski and Chilausky, 1980].

A domain is associated with each attribute used to describe examples. The domain indicates the values the attribute may assume. The values in a domain may be unordered (or nominal), linearly ordered, or hierarchically structured (see Appendix A for a summary of terminology). One of the attributes, called the *classification attribute*, is used to indicate the class to which an example belongs.

This thesis describes a program called "ExceL" (for *Exc*eption *L*earning) which deals with a number of the weaknesses of previous systems. Most notably, the system has the ability to learn rules which have exceptions. The ability to allow exceptions in inductively learned rules is important when the training data is noisy because simpler rules may be generated with little or no loss in rule accuracy, as shown in Section 4.4. It is also a necessary capability for generating rules with *unless* conditions. Using a *main* rule with an *unless* condition is one way to form a rule with multiple conditions that are ordered according to their utility. A rule condition has high utility if it is satisfied relatively frequently. In this form of rule, the main rule tends to be satisfied much more frequently than the *unless* condition. This makes it possible to allow a trade off between speed and precision in an inference system which uses these rules. For example, we may have a rule which states:

If	I turn the ignition key
Then	the car will start
Unless	the car is out of gas or the battery is dead.

The main rule (the *If* part) can be used alone for rapid, low cost reasoning, but with somewhat less confidence than if the unless part of the rule had also been tested. A method for learning rules with

exceptions and unless conditions from examples is discussed in Section 2.3. Some attention is also given
methods for using these rules for deductive inference in Section 2.4.

Induction is a necessarily error prone process. It is falsity preserving rather than truth preservin
That is, except in finite domains, it is not possible to prove a hypothesis generated by induction to be tru
but it is possible to prove it false by finding a counter–example. Thus, a hypothesis generated by indu
tion can rarely be used with 100% confidence. The problem of assigning a confidence level to an indu
tively generated hypothesis is discussed in Section 2.3.2.

Incremental learning is the process of modifying rules so that they are kept consistent with trainin
examples that trickle in over a period time. This capability is important for knowledge base maintenanc
and for "closed loop" learning systems where feedback about rule performance on testing examples is use
to generate new training examples. One approach is to recreate the set of rules from scratch each time
This works, but is computationally costly and has the added problem that the rules may change drasti
cally, confusing the user. A better approach is to generalise or specialise the current rules as necessary t
maintain consistency. An elegant approach to providing both generalization and specialization operator
using a procedure for finding a rule which discriminates one set of examples from another (i.e. a *coverin*
procedure) is discussed in Section 2.3.4.

A conflict exists when a training example leads to mutually exclusive decisions. This is a problem fo
inductive learning systems because it is simply not possible to discriminate between identical objects. A
preprocessor has been provided for ExceL which resolves conflicts in the training data. The conflict han-
dling techniques provided are described in Section 2.2.

Previous learning systems, especially those based on A^q, have been computationally costly to use on
large problems. The efficiency problem was serious enough that considerable research effort was devoted to
techniques for reducing the size of data sets both in the number of training examples used [Cramm, 1983],
and the number of attributes used in describing examples [Baim, 1984]. During the course of the current
work, several techniques were discovered for markedly improving the efficiency of inductive learning for
systems such as ExceL and A^q. These techniques are discussed in Section 2.5.

so given to

preserving.
to be true,
by induc-
an induc-

h training
intenance
les is used
each time.
ge drasti-
cessary to
operators
a *covering*

oblem for
bjects. A
flict han-

to use on
evoted to
m, 1983],
e current
rning for

1.3. Synopsis

This report discusses various aspects of the problem of learning approximate decision rules from imperfect data and using these rules in a resource limited environment. Section 2 describes in a general way the methods used to accomplish the stated goals. Section 3 describes the specific algorithms used in an implementation of these methods, and the reasoning behind the choice of methods. Section 4 presents examples of how the system actually performs on sample problems, and an analysis of this performance. Section 5 summarises the results of this thesis and points out directions for future work. Appendix A gives definitions for many of the terms used in this paper. Appendix B is a user's guide for the ExceL program. Appendix C contains listings of the input data, program output and summary information for the experiments described in Section 4.

Readers who are unfamiliar with Michalski's work should start with Appendix A to become familiar with the terminology and notation used in this paper. The casual reader should read Sections 2 and 5 to get a basic idea of the methodology, and skim the examples in Section 4.

2. DESCRIPTION OF THE METHODOLOGY

Learning decision rules from examples is an incremental process when either incomplete informati
is available at the time of initial rule formation or the environment is dynamic so that the decision ru
must be continually modified to agree with new conditions in the real world. A learning process is *clo*
loop when feedback about performance is used to generate new training examples. Figure 1 shows t
steps involved in a closed loop decision rule learning cycle. Training examples which have been placed
decision classes by an expert are provided to the system. Background rules are used to add new attribut
to training examples with values that are derived from the values of given attributes. The conflict handl
checks for conflicting training examples and makes appropriate modifications to the data. Next, the ru

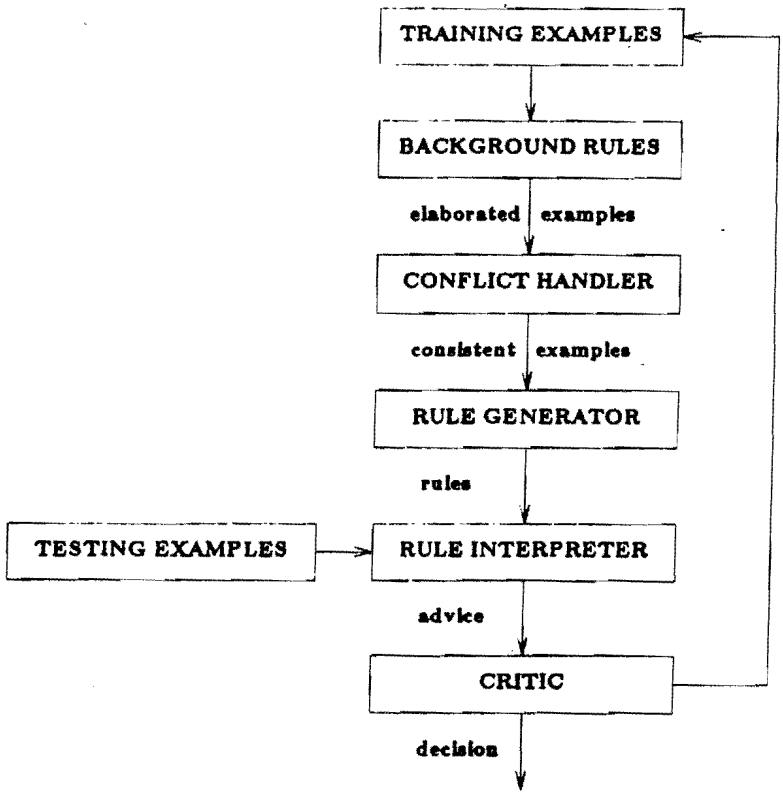

Figure 1. Decision rule learning cycle.

generator induces decision rules from the modified set of examples. The rule interpreter takes the set of decision rules and a testing example and produces a decision, which is presented as advice to a critic. In some situations the critic will be a human expert who has final say about the decision. In other situations the critic will be a component of the computer program. If the advice given to the critic is wrong, the testing example with the correct decision may be recycled as a training example so that the set of rules may be corrected.

2.1. Background Rules

Background rules are used to add or replace attribute–value pairs (selectors) in examples. The value of the new selector will be functionally dependent on the values of given selectors. A background rule consists of three parts:

formula arrow condition.

The *condition* is a conjunction of selectors which must be satisfied by an example for the rule to be applied to it. The *formula* contains the new variables and formulas for computing their values. The *arrow* indicates whether the rule will be used to add (<<—) or replace (<—) selectors.

For example, a background rule for *adding* an area attribute to rectangular objects is written as:

[area := length * width] <<— [shape = rectangle].

If area turns out to be an important attribute for discriminating between the given classes, then much more concise rules may be generated. Applying the above rule to the example:

[class = success][shape = rectangle][length = 5][width = 7]

yields:

[class = success][shape = rectangle][length = 5][width = 7][area = 35].

A rule for *replacing* a numerical area by a qualitative measure of size is written as:

[size := big] <— [area = 30 .. 50].

Applying this rule in turn yields:

[class = success][shape = rectangle][length = 5][width = 7][size = big].

The concept of background rules as described here was first implemented in the INDUCE program for learning structural descriptions from examples [Hoff, Michalski, and Stepp 1983]. A forward chaining process is used to match the conditions of background rules to examples and perform the modifications to training examples.

2.2. Conflict Handling

A conflict exists when training examples with equal values for corresponding attributes occur in more than one decision class. This presents a problem because the inductive learning algorithm used expects the training example sets for different decision classes to be disjoint. It is simply not possible to find a rule for discriminating between identical objects. A conflict may occur because:

(1) The data is noisy – either attributes have been assigned incorrect values or an example has been placed in the wrong class. The first situation may happen when imprecise measurements are used. The second situation may happen when the decision is so difficult that even the human experts do not exhibit perfect performance.

(2) The attributes used provide insufficient information for making the desired discrimination. For example, when discriminating between classes of chemical compounds, *features* of atomic elements and their *relations* may be more relevant to making the discrimination than the *names* of the atomic elements in a compound, since structurally different compounds may contain the same set of elements.

(3) The training example represents a situation where two decisions hold. For example, in a fault diagnosis problem two faults may occur simultaneously, so it may be desirable allow decision rules to overlap. Thus, multiple decisions could be triggered for some testing examples.

The human expert must decide what semantics are to be assigned to the data. The expert should know whether there is likely to be noise in the data, which attributes are necessary for making a discrimination, and whether multiple decisions are to be allowed. The expert can direct the system to behave in one of the following ways when a conflict is encountered:

(1) *Ask* the user. This option is chosen when the data is required to be consistent but is not known to be. A conflict would indicate noise, an inadequate set of attributes, or an inaccurate classification by an expert.

(2) *Drop* conflicting examples from all classes involved in a conflict. This option should be chosen when the data is known to be noisy, and the noise is evenly distributed across all attributes, or occurs in the classification attribute.

(3) *Assign* an example which causes a conflict only to the class where it occurs the most *frequently*. It is sometimes useful to associate frequency data with training examples, and duplicate examples are allowed, so in some cases it is desirable to use this information for conflict handling. This option may be chosen when the data is known to be noisy, but there is a relatively high probability that training examples will be assigned to the correct class.

(4) *Keep* conflicting examples in all classes involved in a conflict. This option is chosen when multiple decisions for an example are expressly allowed. This differs from doing nothing in that modifications are actually made to the training example sets used by the rule induction algorithm.

(5) *Do nothing*. This option may be chosen when the data is known to be consistent and the user does not want to waste processing time checking for conflicts.

These methods of conflict handling are believed to be adequate to handle most situations. The expert is given explicit control, yet relieved of the chore of manually making the example sets consistent. At this point, implemented systems are not capable of storing or using enough knowledge about the real world to perform the task of determining whether or not a given data set is noisy. Nor is it possible to automatically determine whether multiple decisions should be allowed for examples. Once conflicts in the training examples are resolved, the consistent set of training examples may be passed on to the rule generator.

2.3. Learning Rules with Exceptions

The task of the rule generator is to to form a rule describing each decision class. This rule may then be used to distinguish examples of one class from examples of any other class. The rule may be approximate – it may have exceptions. There are two kinds of exceptions: examples which should be covered by a rule for a particular class but are not, called *positive exceptions*, and examples which are covered by a rule but should not be, called *negative exceptions*. Figure 2 illustrates a rule with positive and negative exceptions.

Rule:	*If*	I turn the ignition key
	Then	the car starts.
Positive Exception:	*If*	I hot–wire the ignition, and the gas tank is full, and the battery is charged
	Then	the car starts.
Negative Exception:	*If*	I turn the ignition key, and the gas tank is empty, and the battery is charged
	Then	the car does not start.

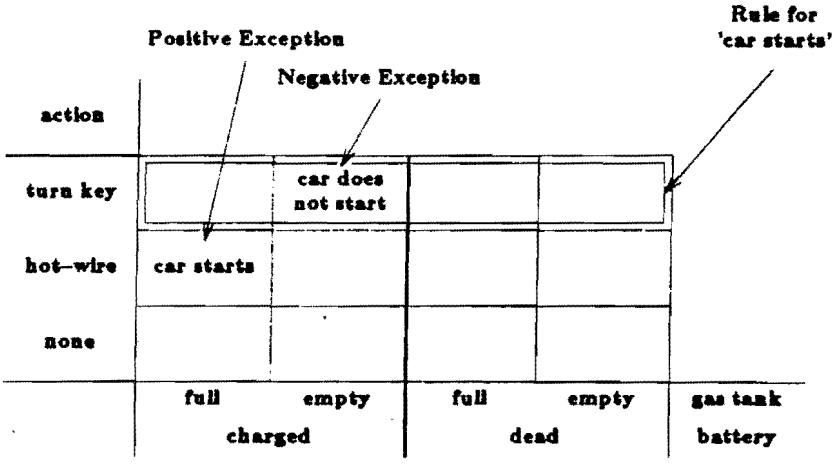

Figure 2. A rule with positive and negative exceptions.

Positive exceptions are of interest in two cases: when the data is noisy, and when examples have unique names. If the data is noisy, generated positive exceptions may be dropped from further consideration. If the data is not noisy, and unique names are provided for examples, the positive exceptions may be enumerated easily using their names. Otherwise, a rule which covers all positive examples should be used.

Negative exceptions are also of interest in two cases: when the data is noisy, and for generating rules with *unless* conditions. As for positive exceptions, when the data is noisy the generated negative exceptions are dropped from further consideration. If the data is not noisy, an unless condition can be used to summarise the negative exceptions found for a rule.

2.3.1. Characteristics of Rules with Unless Conditions

The form of a rule with an unless condition (also referred to as a *censor* [Winston, 1983; Michalski and Winston, 1985]) is shown in Figure 3. Formula 1 is the normal form for a rule with an unless condition where D is the *decision*, P is the *premise*, C is the *censor*, the symbol $\llcorner$ means *unless*, and γ represents the *confidence* in the decision when the premise is satisfied but the censor is untested. There are two types of censors – active and passive. Active censors only apply to logical decision rules and represent a condition which is mutually exclusive with the decision. If an active censor is satisfied, the negation of the decision is known to be true. A logically equivalent form for an active censor is shown in Formula 2. Passive censors apply to all types of production rules and represent a condition under which the decision cannot be triggered. Formula 3 gives a logically equivalent form for a passive censor. If the censor is

Formula 1. $D \Leftarrow P \llcorner C : \gamma$

Formula 2. $(D \otimes C) \Leftarrow P$

Formula 3. $(D \vee C) \Leftarrow P$

Figure 3. The form of rules with unless conditions.

tested and fails to be satisfied, the confidence in the decision is 1 (certainty). For a discussion of the development of rules of this form and additional semantic considerations which are not dealt with here see [Michalski and Winston, 1985].

A fundamental goal behind the creation of rules with unless conditions is that they provide a technique for implementing a *variable-precision logic*. That is, it is possible to specify guidelines for satisfactory confidence levels and resource utilisation, and modify the way the rules are used to meet these guidelines. To meet this goal, the parts of a rule must have certain properties:

(1) The decision must hold with a high degree of confidence for a majority of the cases when the premise is true. We should be able to do reasoning with only the premises of rules, ignoring unless conditions, and be able to reach conclusions with a reasonably high confidence.

(2) The *unless* condition must hold with a high degree of confidence for a small number of cases when the premise is satisfied but the decision is false (active censor) or unknown (passive censor). Note that if the unless condition holds for a large number of cases when premise is satisfied, then the confidence γ must be low.

More formally, consider some rule R with confidence γ. Let Ω be the universe of all training examples, Ω_P be all examples such that the premise of R holds, Ω_{PD} be all examples such that both the premise and decision of R hold, and Ω_{PC} be all examples such that both the premise and censor of R hold. Given total knowledge we have

$$\gamma = \gamma_1 = \frac{|\Omega_{PD}|}{|\Omega_P|}.$$

If we let

$$\gamma_2 = \frac{|\Omega_{PC}|}{|\Omega_P|}$$

then

$$\gamma_1 + \gamma_2 = 1.$$

An unless condition should only be generated when

$$\gamma_1 > \frac{1}{2}.$$

That is, the main rule should have a confidence of greater than 50% without testing the unless condition. Usually a much higher confidence level will be needed to do useful reasoning.

2.3.2. Assigning a Confidence Level

A difficult problem is deciding what confidence level should be given to a rule acquired by induction. The confidence level assigned to a rule should reflect the probability that the rule will give the correct decision, assuming that the censor (if any) is untested. If the set of training examples is exhaustive as assumed above, then the probability that a rule gives the correct decision for a particular example is:

$$\frac{|E_p|}{|E_p|+|E_n|}$$

where E_p represents the set of observed positive examples covered by the rule, and E_n represents the set of observed negative examples covered by the rule. This expression is equivalent to the one given above for γ_1 since both expressions represent the ratio of the number of covered positive examples to the total number of covered examples, positive or negative.

When the training data used is a subset of all possible training examples, the accuracy of inductively generated rules depends on the percentage of all training examples which are observed, and the complexity of a correct decision rule [Quinlan, 1983a]. It is often not possible to determine a priori the total number of possible examples since certain combinations of values may be impossible, and an event space containing real number valued attributes is infinite. Also, it is not possible to know a priori the complexity of a correct decision. But, as Quinlan points out, for large numbers of training examples (several thousand or more) the accuracy which can be achieved by inductive learning is relatively independent of these factors and is quite high (over 99%).

The confidence measure currently used is the above formula modified to take into account confidence levels associated with individual examples:

$$\frac{\sum_i C_{p_i}}{|E_p| + |E_n|}$$

where C_{p_i} is the confidence associated with the *i-th* positive training example. The confidence level p
duced is, in general, unrealistically high since the measure only takes observed examples into accou
That is, it is assumed that a sufficiently great number of training examples have been provided so that it
possible to generate rules which are highly accurate. Since rules may be refined to agree with new
observed examples this is not a great shortcoming.

2.3.3. A Learning Technique

The ExceL algorithm learns class covers from examples, where a class cover has the form:

decision $\Leftarrow$ complex $\vee$ complex $\vee$ ... $\vee$ complex.

The left hand side is a single selector which specifies the decision class. The right hand side is a disjun
tion of *annotated* VL_1 complexes. Annotation gives meta-information about the rule. The technique us
for learning is a branch and bound search through a space of conjunctive descriptions, progressing fro

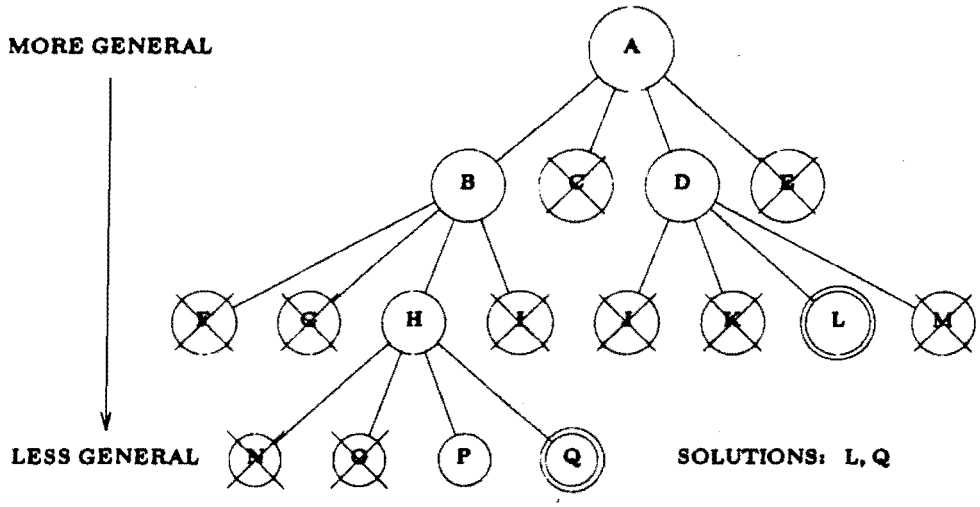

Figure 4. Branch and bound search strategy used in ExceL.

more general to less general descriptions as shown in Figure 4. In Figure 4, each node represents a candidate description. The initial description for a decision class is based on a "boundary" complex (node A), which usually covers the entire event space. If a description covers any negative examples, a number of alternative specializations of the description are generated which do not cover a selected negative example. A desirable subset of these descriptions is selected at each "bound" stage according to predefined criteria. When the confidence of a description becomes high enough, it is added to a list of solutions. When enough solutions have been collected, the best one is chosen to become part of the cover. Since a single complex may fail to cover enough of the examples for a class, the process is repeated, yielding a description in disjunctive normal form (DNF).

As a simple example, we might have a set of training examples describing when a car will start, and when it will not start such as:

frequency	car	action	gas_tank	battery
100	starts	turn_key	filled	charged
1	starts	hot_wire	filled	charged
1	does_not_start	turn_key	empty	charged
1	does_not_start	turn_key	filled	dead
100	does_not_start	none	filled	charged
1	does_not_start	none	empty	charged
1	does_not_start	none	filled	dead
1	does_not_start	hot_wire	filled	dead
1	does_not_start	hot_wire	empty	charged

Each row represents a training example that occurs with the relative frequency indicated in the first column. Each column heading indicates the name of an attribute and the entries below it indicate the value for that attribute in each training example. The attribute "car" determines the decision class for each example. When directed to produce exact rules, the learning algorithm generates these rules from the above examples:

$$[car = does\_not\_start] \Leftarrow$$
$$[action = none] : (1.0, 102, 102, 0) \vee$$
$$[gas\_tank = empty] : (1.0, 2, 3, 0) \vee$$
$$[battery = dead] : (1.0, 2, 3, 0)$$

$$[car = starts] \Leftarrow$$
$$[action = turn\_key \vee hot\_wire][gas\_tank = filled][battery = charged] : (1.0, 101, 101, 0)$$

The first rule states that a car will not start if no action is taken, or the gas tank is empty, or the battery is dead. The second rule states that a car will start if key is turned or the ignition is hot–wired, and the gas tank has gas in it, and the battery is charged. In this case, the rules are logical complements of each other. The parameters to the right of each complex specify, in order, the *confidence* with which the associated decision can be made if the complex is satisfied and the censor (if any) is untested, the number of positive training examples which are *newly covered* with respect to the order of complexes in the rule, the *total number of covered positive training examples*, and the *number of covered negative training examples* for the complex. This is an exact rule – no exceptions were allowed. The system can also be directed to allow exceptions to rules.

Negative exceptions are recognised as those negative examples which remain covered after a complex has been specialized enough so that the majority of negative training examples are no longer covered. The number of negative examples which may remain covered is determined by the confidence that the rule must have. Positive exceptions are recognized as those positive examples which remain uncovered after a majority of the positive examples have been covered by complexes of high utility, or which are covered only by complexes of low utility. The utility of a complex is defined as the fraction of all positive examples covered by the complex. Thus, a gestalt criterion of overall rule quality is used to identify exceptions. Isolating exceptions may be viewed as a three class clustering problem. The concept of using overall rule quality to guide clustering is also used in the CLUSTER/2 algorithm [Stepp, 1984].

For example, if we allow a minimum confidence of 0.9, and a minimum utility of 0.1, then the system generates these rules from the above data:

[car = does_not_start] ⇐
    [action ≠ turn_key] : (0.99, 104, 104, 1)

Negative Exceptions:
    [action = hot_wire][gas_tank = filled][battery = charged]

Positive Exceptions:
    [action = turn_key][gas_tank = empty][battery = charged]
    [action = turn_key][gas_tank = filled][battery = dead]

[car = starts] ⇐
    [action = turn_key] : (0.98, 100, 100, 2)

Negative Exceptions:
    [action = turn_key][gas_tank = empty][battery = charged]
    [action = turn_key][gas_tank = filled][battery = dead]

Positive Exceptions:
    [action = hot_wire][gas_tank = filled][battery = charged]

The exceptions are chosen from among the training examples and are not annotated. Obviously, these approximate rules are much simpler than the corresponding exact rules if the exceptions are ignored, and they will work correctly most of the time.

An unless condition can be generated by covering the negative exceptions of a complex against the positive training examples covered by the complex. This should be done only when the domain expert has determined that the negative exceptions are valid training examples. From the above data the system produces:

[car = does_not_start] ⇐
    [action ≠ turn_key] : (0.99, 104, 104, 1) ⌊
        [action = hot_wire][gas_tank = filled][battery = charged] : (1.0, 1, 1, 0)

[car = starts] ⇐
    [action = turn_key] : (0.98, 100, 100, 2) ⌊
        [gas_tank = empty] : (1.0, 1, 1, 0) ∨
        [battery = dead] : (1.0, 1, 1, 0)

The positive exceptions remain the same as shown for the preceding example. In the rule for a car not starting, it is not possible to generate an unless condition which is simpler than the negative exception it

was formed from because the training examples used restrict generalisation. In the rule for a car starting, summarising the negative exceptions in an unless condition gives the rule in a form which people seem to find more desirable than the purely conjunctive form of the exact rule above. The program input and output for these examples is given in Appendix C.1.

2.3.4. Incremental Learning

Incremental learning allows rules to be modified with a minimum of effort when new training examples become available. The basic operations needed for incremental learning are a classification operation, a generalization operation and a specialization operation [Becker, 1985b]. The specialisation step should precede the generalisation step since, if covers are required to be disjoint, a covered negative example must be uncovered by an incorrect rule before it can be covered by the correct rule. In order to ensure consistency with previously observed examples, it is necessary to keep a record of them.

Classification involves determining which rules cover a new training example and updating the records associated with each decision class. Both generalisation and specialisation can be implemented using the covering operation described above. Generalising a class cover is done as described above, except that the complexes of the current class cover along with any uncovered positive examples are used as the positive training examples for the class. The key observation needed for using the covering operation for specialisation is to recognise that the initial description (the *boundary*) need not be the entire event space, but can be some subset of the event space. Specialising a complex is done by covering the covered positive examples against the covered negative examples, using the complex as the boundary.

This technique has been applied by Bob Reinke as a modification to A^q. Reinke found that descriptions generated by incremental learning tend to be slightly more complicated than those generated by single step learning, but that less total CPU time is required for the induction process, and that rule accuracy is not affected much [Reinke, 1984].

The use of unless conditions in rules provides more flexibility in the incremental learning process. Specialisation need not be done if a covered negative example is already covered by the unless condition of

a complex and the confidence of the complex is sufficiently high.

2.4. Interpreting Rules with Unless Conditions

Although this thesis is primarily concerned with learning, some attention must be given to how rules with unless conditions can be applied. Production systems may be forward chaining, backward chaining, or bi-directional [Nilsson, 1980]. Rules with unless conditions may be used in any of these systems, but the manner in which they are used varies. During backward chaining, the system acquires new information from the user. The system asks the user questions which are least costly to the user, and still achieve a certain level of confidence. During forward chaining, all information needed to fire a rule is assumed to be available, so there is no cost associated with acquiring information. Unless conditions are evaluated only when the rule cannot otherwise be used with a high enough level confidence. Thus, the level of confidence required determines the amount of reasoning which will be done by the system.

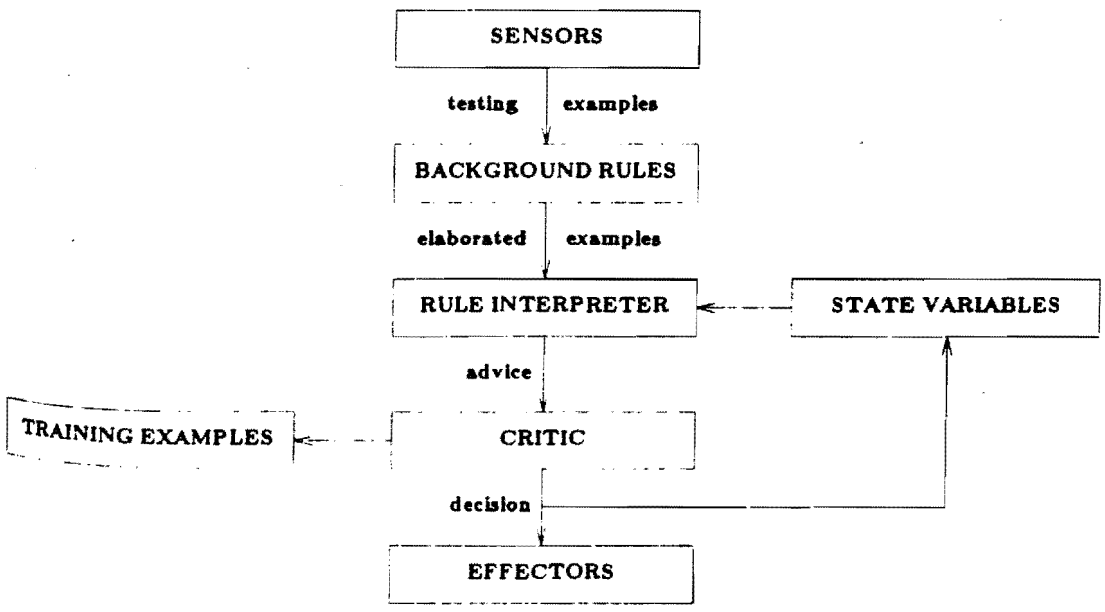

Figure 5. Rule interpretation cycle.

Figure 5 illustrates a system which is primarily forward chaining. This is the same system as shown in Figure 1, but with a different focus of attention. The cycle proceeds as follows. Sensors are used to acquire information from the environment which serves as a testing example. Background rules are applied to elaborate the testing example. The input information and a VL_1 complex representing the internal state (or short term memory) of the system are sent to the rule interpreter which decides what actions to do. Conflict resolution is not done because the learning system is expected to ensure consistency of the rules. *All* rules which are selected are fired in parallel. All input information and the set of actions selected by the rule interpreter are passed to the critic, which may be a human or a program module. The critic returns the *correct* set of actions, which may or may not be the same as those selected by the system. The correct actions are then triggered and the internal state of the system is updated. Backward chaining may be invoked by the action part of a forward chaining rule to fill in unknown values by querying the user. If the actions selected by the system do not agree with those given by the critic, one or more training examples are created and sent to the learning system which updates the set of rules.

This scheme is just one of many possible schemes for making use of rules with unless conditions in an inference system. Winston describes a system in which unlimited effort is used in evaluating the main rule but only a single inference step is used to evaluate unless conditions in [Winston, 1983]. It may be beneficial to use different evaluation schemes depending on the meaning associated with the unless condition. For example, in the car-starting rule the unless conditions describe causal preconditions. In this case it would be useful to treat the unless conditions as "things to check" if the action of turning the key fails to produce the desired effect.

2.5. Performance Considerations in Learning

The problem of learning class descriptions from examples is treated here as a heuristic search process. Better results can be achieved for a given amount of computational effort if the search process is made more efficient, enabling the investigation of a greater fraction of the search space. Two ways to improve search are intelligent pruning and the use of better heuristics. Also, performance may be improved by taking advantage of storage versus computation time trade-offs. All of these techniques are

used in ExceL to improve performance.

As previously stated, the learning algorithm uses a branch and bound search in a conjunctive description space to create and select descriptions. In learning, it is important to focus on inconsistencies and borderline cases. One form of intelligent pruning is based on the observation that if a negative example is not covered by a particular description, the example may be removed from further consideration. When a conjunctive description covers no negative examples (or satisfies a confidence level criteria) it is "done", since further specialization will not improve it. It is removed from the set of candidates and added to the set of solutions.

A good discriminant description is brief, covers all of the positive examples for a class, and covers no negative examples. A good *approximation* to a good discriminant description is also brief, covers a large proportion of the positive examples for a class, and a small proportion of the negative examples. Thus, there should be a heuristic which selects this type of description. Previous systems have provided evaluation functions for counting the number of positive and negative examples covered by a description, but these are not the best possible measures of quality. An effective measure of rule quality is:

$$P^*(description) = \frac{p}{P} - \frac{n}{N}$$

where p is the number of covered positive examples,
P is the total number of positive examples,
n is the number of covered negative examples,
and, N is the total number of negative examples.

Not only are the generated descriptions usually more concise when this measure is used instead of counts of covered examples, but the computation time for cover generation is also improved, as shown in Section 4.3. P^* is a measure of the relevance a *description*, which may be a selector, a complex, or a DNF rule, for making a discrimination between two classes. A P^* value of +1 means a description covers only positive examples, and a P^* value of -1 means it covers only negative examples.

P^* is closely related to the Promise measure of *attribute* relevance developed by Baim [Baim, 1984]. If an attribute has a Promise of 1, it can be used alone to discriminate between a set of decision classes. A

Promise of 0 means that an attribute provides no information for making a discrimination. Promise may be computed from the relative frequencies of occurence of the values of attributes in training examples according to the formula:

$$Promise\ (A) = \frac{\sum_{v} max_{c} \left(R_{cv} \right) - 1}{m - 1}$$

where A is the attribute being tested,
v is a value of attribute A,
c is a class,
R_{cv} is the relative frequency of v in the examples of class c,
and m is the number of classes.

This formula for Promise is developed in [Becker, 1985b] and is equivalent to the formula developed in [Baim, 1984]. As an illustration of the correlation between Promise and P^*, consider this table of relative frequences for the values a, b, c and d of some attribute V1, in classes A and B:

	V1			
Class	a	b	c	d
A	1/10	5/10	0/10	4/10
B	7/12	1/12	3/12	1/12
max	7/12	5/10	3/12	4/10

$$Promise\ (V1) = \frac{(7/12 + 5/10 + 3/12 + 4/10) - 1}{2 - 1} = 0.7333$$

Given a selector constructed to maximise the value of P^* for one of the classes, the value of P^* for the selector is *equal* to the Promise value for the attribute (this relation holds only when there are two decision classes). For example, the optimal selector for class A in the above table is

$$[V1 = b \vee d].$$

P^* for this selector is

$$P^* = \frac{5 + 4}{10} - \frac{1 + 1}{12} = 0.7333$$

The same result is obtained from the optimal selector for class B.

Evaluating heuristics can involve considerable computation if the data needed is not readily available. Two general types of information are used for evaluating descriptions: information which is derived from the description itself such as the number of literals, and information which relates the description to the training examples. Information about the description itself is generally easy to compute. Information about how the description is related to training examples can be expensive to compute if done improperly. For example, in existing implementations of the A^q algorithm, the program must compare a description with each example one at a time to determine how many positive or negative training examples are covered. In ExceL, examples are indexed into a data base that allows the system to determine the set of covered examples for a complex with a computation speed that is independent of the number of examples. Also, three sets of examples are stored with each complex in a rule – the set of covered positive examples, the set of covered negative examples, and the set of covered positive examples which are not covered by a previously generated complex in the rule.

3. DESCRIPTION OF THE IMPLEMENTATION

This section provides a more detailed look at the algorithms used in the implementation of the system described in this thesis. All code is written in FRANZ LISP [Foderaro, Sklower, and Layer, 1983] under Unix 4.2bsd, and makes extensive use of a macro package described in [Becker, 1985a]. The source code consists of the following files:

File	Lines	Bytes	Description
cover.l	33	977	Bootstrap loader for the system
excel.l	1051	35143	Induction algorithms
excer.l	142	4591	Deduction algorithms
backgrd.l	498	15030	Background rule parser and applier
dataset.l	850	27941	Data set management routines
sets.l	2419	74803	Generic set operations
dbvl.l	745	22282	VL_1 data base operations
vl1.l	826	25297	VL_1 selector and complex operations
parse.l	1046	31423	Data driven parser
arith.g	99	2828	Grammar for background rules
textio.l	457	14208	User interface
TOTAL	8166	254323	

Certain parts of the system, in particular the set operations and VL_1 operations, contain many functions which are not actually used by this system but are provided so that these packages may be used as components in other systems. The data driven parser is described in [Becker, 1985c].

The basic steps involved in processing a data set are as follows:

(1) Training events are indexed into a data base.

(2) Background rules are applied to the events, modifying or adding selectors.

(3) Classes are defined by the domain of the classification attribute. A classification predicate is created for each class.

(4) A record is created for storing the information associated with each decision class. The sets of positive and negative examples for each decision class are stored in these records.

(5) Conflicts are handled in one of the available modes.

(6) A rule is generated for each class, either in a single step or incrementally.

(7) The rule interpreter is optionally applied to the rules.

The first four steps consist of parsing and bookkeeping operations. These will not be described in detail. The last three steps will be described and analysed further.

8.1. Conflict Handling

Conflict handling is a relatively simple process. As previously discussed (Section 2.2), one of five options may be selected. If the user chooses to have no conflict handling done, the procedure is not invoked. The algorithm is given in Figure 6. In this algorithm, the parameter *eventset* is the set of events being tested for conflicts. The parameter *db* is a VL_1 data base in which the events are indexed. The parameter *classdescriptions* is a list of descriptions for the decision classes in the current data set. A class description is a data structure which stores all information relevant to a particular class, including training

```
HANDLECONFLICTS (eventset, db, classdescriptions, mode)

  repeat
      event := next (eventset)
      equivset := nondisjoint (event, db)
      classes := getclasses (equivset, classdescriptions)
      if (cardinality (classes) > 1) then
          case mode of
            ask:  Print (classes, "Which class is correct?")
                  keeper := (read)
                  negevents(keeper) := negevents(keeper) - equivset
                  equivset := equivset - posevents(keeper)
                  for class in (classes - keeper) do
                     posevents(class) := posevents(class) - equivset
                     negevents(class) := negevents(class) - equivset
            drop: for class in classes do
                     posevents(class) := posevents(class) - equivset
                     negevents(class) := negevents(class) - equivset
            keep: for class in classes do
                     negevents(class) := negevents(class) - equivset
            max:  keeper := getmax (equivset, classes, gamma)
                  negevents(keeper) := negevents(keeper) - equivset
                  equivset := equivset - posevents(keeper)
                  for class in (classes - keeper) do
                     posevents(class) := posevents(class) - equivset
                     negevents(class) := negevents(class) - equivset
          end (* case *)
      end (* if *)
      eventset := eventset - equivset
  until (empty (eventset))
end (* HANDLECONFLICTS *)
```

Figure 6. Conflict handling algorithm.

events and the class cover once it is generated. The parameter *mode* is the conflict handling mode. The function *next* returns the first event present in a set of events. The function *nondisjoint* returns the set of events in the data base which overlap with the given event. The function *getclasses* returns the set of class descriptions associated with the events involved in a conflict. The functions *posevents* and *negevents* return the positive and negative training examples associated with a class, respectively. And, the function *getmax* returns the class where the conflict event occurs the most frequently, provided the most frequent occurance is *gamma* times more frequent than the next most frequent occurance.

Note that to *drop* an event from a class involves removing it from both the set of positive examples and the set of negative examples for that class, but to *keep* a conflict event involves removing it only from the sets of negative examples. This allows the learning algorithm to generate covers for different classes which cover the same event.

3.2. Learning Rules with Exceptions

The technique used here for learning rules with exceptions resembles the A^q algorithm in that both solve the covering problem by generating VL_1 descriptions in disjunctive normal form. The differences between the algorithms are substantial. The learning algorithms used in ExceL will be described in detail, and differences between the algorithms used in ExceL and A^q will be discussed.

Figure 7 gives the covering algorithm used in ExceL. The purpose of this algorithm is to find a disjunction of complexes which cover most of the positive training examples and few of the negative training examples for a decision class. The parameters *posexamples* and *negexamples* are the positive and negative examples for the decision class which is being covered. To form covers for several classes, each class is covered in turn using the examples for the class being covered as positive examples, and the examples from all other classes as negative examples. The degrees to which positive and negative exceptions are allowed are controlled by the *utility* and *confidence* parameters respectively. These parameters are used as thresholds. The utility of a complex is the fraction of all positive examples that it covers. The confidence is as defined in Section 2.3.2. The *boundary* parameter is a VL_1 complex which specifies the most general

```
COVER (utility, confidence, posexamples, negexamples, boundary, LEF)

  uncovered := posexamples
  totalpos  := cardinality (posexamples)
  totalneg  := cardinality (negexamples)

  repeat
    refu := refunion (uncovered)
    put-annotation (boundary, uncovered, posexamples, negexamples)
    star := ADG (confidence, refu, boundary, LEF)
    bestcomp := bestcomplex (star, LEF)
    bestcomp := trim (bestcomp)
    uncovered := uncovered - coveredpos (bestcomp)
    if ( util(bestcomp) > utility)
      then
          cover := cover ∪ bestcomp
          poscovered := poscovered ∪ coveredpos (bestcomp)
      end (* if *)

  until (uncovered / totalpos  <  utility)

return (cover, (posexamples - poscovered))
end (* COVER *)
```

Figure 7. Cover generation algorithm.

allowed description. This is used for incremental learning, which was described in Section 2.3.4 The algorithm returns a cover for the class described by the sets of positive and negative examples which has the properties that each complex in the cover has a utility greater than the utility threshold, a confidence greater than or equal to the confidence threshold, and is covered by the boundary complex. A utility threshold of 0.0 and a confidence threshold of 1.0 will cause the algorithm to produce covers with no exceptions. Also, the algorithm returns the set of uncovered positive examples, i.e. the positive exceptions. Negative exceptions are recorded as annotation on individual complexes in the cover.

The process involves generating complexes which cover some fraction of the remaining uncovered positive training examples until most are covered. During each major cycle, first the refunion of the remaining uncovered positive events is found. The refunion of a set of events is a complex which is constructed by taking the union of the values for each attribute, as shown in Figure 8. Note that a selector is

Event 1:	[color = red][shape = octagon][reflective = yes]
Event 2:	[color = white][shape = square][reflective = no]
Event 3:	[color = yellow][shape = triangle][reflective = yes]
Refunion({1, 2, 3}):	[color = red ∨ white ∨ yellow][shape = octagon ∨ square ∨ triangle]

Figure 8. An example of applying the refunion operation.

omitted if all values from the domain of the attribute are present. Next, the sets of uncovered positive, all positive, and negative examples are recorded as annotation on the boundary complex. ADG (Figure 9, described below) is then called to produce a set of descriptions for the uncovered positive examples. These descriptions will all be specializations of the given boundary complex. A single complex is selected as the best description according to user defined quality criteria given in a *Lexicographic Evaluation Function with Tolerances* (LEF - see Appendix A). The LEF uses quality criteria such as P^*, the total cost of all variables in a complex, and the average user assigned weight (relevance) of all variables in a complex, where cost and weight are quantities assigned by the domain expert. Next, positive examples covered by the best complex are removed from the set of uncovered examples. If the utility of the best complex is high enough, it is added to the cover. This process continues until too few uncovered positive examples remain, as determined by the utility threshold.

The complexes in the cover may also be *trimmed*. The purpose of trimming is to simplify rules and reduce overgeneralization by removing values from selectors in a complex when they are not needed in order to cover the positive examples actually covered by the complex. Trimming may been done with respect to the set of positive examples which are *uniquely* covered by the complex, or with respect to the set of *all* positive examples covered by the complex. The former will usually result in greater simplification than the latter, but can change the utility of a complex.

Figure 9 gives the alternative description generation (ADG) algorithm. The purpose of this algorithm is to find a set of alternative conjunctive descriptions which cover a large proportion of the set of

```
ADG (confidence, refu, boundary, LEF, $solutions, $probe)

  probe := 0
  star  := boundary

  repeat
     probe := probe + 1
     star := selectbest (star, maxstar, LEF)
     newstar := empty

     for complex in star do
        if ( conf(complex) > confidence)
           then
              solutions := solutions ∪ complex
              probe := 0
           else
              negevent := Getevent (coveredneg(complex))
              negcomps := Subtract (refu, negevent)
              newstar := newstar ∪ Multiply (complex, negcomps)
           end (* if *)
     star := newstar

  until ((cardinality(solutions) > $solutions)
        or (and (cardinality(solutions) > 0)(probe > $probe)))

  return solutions
end (* ADG *)
```

Figure 9. Alternative description generation algorithm.

uncovered positive training events and a *confidence* greater than the given threshold. The resulting descriptions will be specializations of the given *boundary* complex, non–disjoint with the given refunion complex, and the "best" according to the given *LEF*.

The technique used is a beam (branch and bound) search. During each cycle, the confidence of each complex in the star (set of alternative descriptions) is tested. If the confidence is high enough, the complex is added to the set of solutions. Otherwise, the complex is specialized by selecting some covered negative event, subtracting it from the refunion complex, and finding the intersection the complex with each of the single selector complexes resulting from the subtraction using the *Multiply* function. This yields several new complexes, each of which is disjoint with the selected negative event. The new star is the union of

these newly specialised complexes. A certain maximum number *maxstar* of these descriptions are selected for further processing using the LEF to decide which descriptions to keep. The algorithm terminates when $*solutions* number of solutions have been found, or some solution has been found and $*probe* more cycles have been completed. Since the solutions become progressively more specialised, and hence more complex, the solutions found early on are typically the best.

3.3. Comparison to A^q

A^q implements a technique based on the method of *disjoint stars*, which has been proven to be a quasi–optimal solution to the general covering problem [Michalski and McCormick, 1971]. The task of finding an optimal solution to the general covering problem has been shown to be NP–complete [Hong and Michalski, 1985]. "Quasi–optimal" means that if the algorithm does not return a cover containing the minimal number of complexes, it can return a value indicating what the optimum is. The A^q algorithm works by building stars, where a star is a set of all the maximally general alternative generalisations of some example. The example around which a star is built is called a seed. A major difference between ExceL and A^q is the use of the refunion of positive examples rather than individual seed examples as a starting point for generalisation. Because the method of disjoint stars is not used, ExceL is not known to be quasi–optimal. In practice this does not seem to be important.

The decision to use the refunion of positive examples rather than seed examples was made so that positive exceptions could be selected according to the rule quality criteria. If seeds were used, some example which should be allowed as a positive exception might possibly be selected as a seed, and thus would necessarily be covered by the complex created as a generalisation of it. Another benefit of this approach is that overgeneralisation in intermediate descriptions is not as extreme. A problem with this approach is that the system is *not* good at finding rules for classes which have several distinct clusters of events, and which are best described in terms of a disjunction of disjoint complexes. The problem is overcome by directing the system to accumulate a large number of solutions before picking one to become part of the cover.

ExceL also differs from A^q in the termination conditions that are used. In A^q, the set of complexes in a star are specialized until all negative examples have been uncovered. This is done by processing each negative example in turn whether it is covered or not. In ExceL, large numbers of negative examples may be skipped over in a single cycle, and some negative examples may remain covered. A^q also continues until all positive examples of a class have been covered while Excel may leave some uncovered.

Like A^q, ExceL may be used to generate several different types of covers. If just the training examples are passed to the covering algorithm, then the rules produced may overlap in *don't care* space. These are called *intersecting covers*. If previously generated covers are included with the negative examples, the rules will be disjoint. These are called *disjoint covers*. Covers may also be generated in such a way that only examples in classes which follow a particular class in the given order are treated as negative examples. The result is simpler rules which must be applied in the order generated. These are called *ordered covers*. Both algorithms may be used to generate rules for hierarchical decisions by applying the covering procedure at each level of the hierarchy, and for incremental cover generation.

8.4. The Rule Interpreter

The rule interpreter (Figure 5) is a simple production system based on VL_1 which can be used in both a forward and backward chaining mode. The backward chaining mode is not fully implemented. It is structured as a state machine, where the input information, current state, and sets of actions are all represented as VL_1 complexes. The system is designed to interact with the external world and a critic so that it can produce training examples which may be used by the learning system to modify the set of rules. The general purpose section of the rule interpreter consists of procedures for applying background rules to input complexes, selecting rules to fire, updating the state complex, backward chaining, and interfacing to the input, critic, and effector routines.

The input, critic, and effector routines are domain specific and must be rewritten for each new application. The input routine returns a new complex indicating values received from a set of sensors each time it is called. The sensors may be real world measurement devices or routines which access values in a simu-

lation program. The critic routine is called with the input data which has been elaborated using the given background rules, the internal state, and a complex representing the set of actions selected by the system, and returns a complex representing the *correct* set of actions. The effector routine is called with a complex representing a set of actions, and is expected to perform operations on the environment according to the indicated actions.

3.5. Performance Considerations

Heuristic and algorithmic factors affecting performance have already been considered. Two implementation factors of special importance for obtaining good performance in a learning system based on VL_1 are a flexible package of set operations and an efficient VL_1 data base. These make it possible to efficiently perform operations such as union and intersection on VL_1 descriptions, and to efficiently evaluate heuristics concerned with the relationship between descriptions and training examples.

The set operations allow non-monotonic changes in the members of a set *type* so that sets of complexes, which are created and destroyed during learning, can be represented. Also, all VL_1 types are supported by the set operations package, including nominal, linear, structured, and cyclic.

The data base system is used for classifying events, determining what events are covered by a candidate rule, selecting background rules to apply, and selecting production rules to fire in the rule interpreter. The data base system in the current implementation relies heavily on the set operations package. The operations provided are:

index	*index a complex into the data base*
unindex	*remove a complex from the data base*
covers	*get the set of complexes which are covered by the given complex*
coveredby	*get the set of complexes which cover the given complex*
disjoint	*get the set of complexes which are disjoint with the given complex*
projection	*project the data base onto a subset of the set of attributes*

The structure of the data base is illustrated in Figure 10. The data base contains a subtable for each attribute, and each subtable has a set of complexes for each allowed value of the attribute. A complete secondary index is created for each attribute value. That is, if a complex has a certain value for a certain

attribute, the bit corresponding to the complex is set in the appropriate slot of the appropriate subtable. Lookup is accomplished by performing various boolean operations over the sets of complexes. For the *covers* and *disjoint* lookup operation, the time required is proportional to the number of attribute values present in the probe complex. For the *coveredby* lookup operation, the time required is proportional to the number of attribute values present in the probe complex plus the number of selectors which are declared but not present in the probe complex.

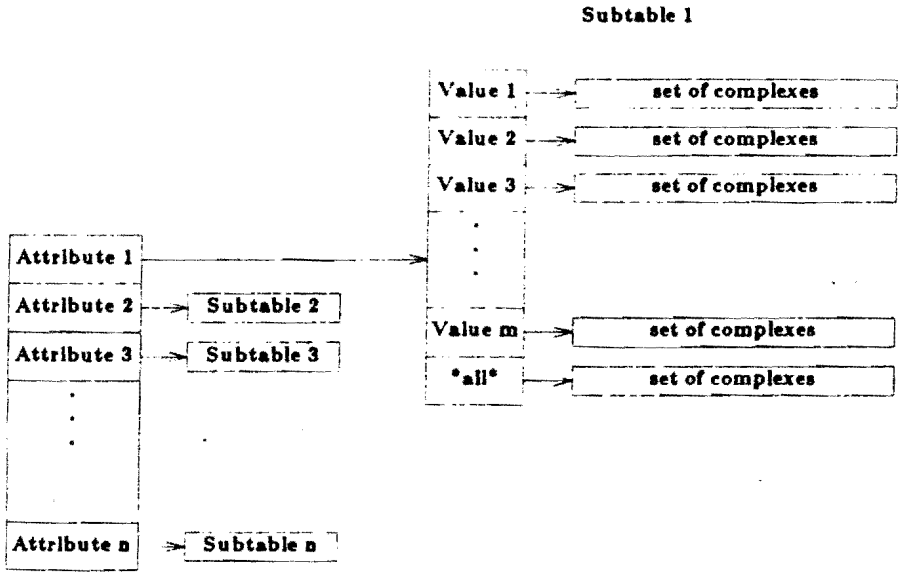

Figure 10. VL_1 Data Base Structure.

4. EXPERIMENTATION AND ANALYSIS

A number of experiments were performed to test various aspects of the ExceL system. First, the system was tested against an implementation of A^q to determine what differences in performance should be expected. Second, the system was run with a different evaluation function to see whether using P^* as a quality criterion has a significant effect. Third, two experiments performed by Quinlan using a version of ID3 to test the effectiveness of inductive learning on noisy data were repeated using ExceL. These experiments were then repeated using approximate decision rules. Finally, an example of a closed loop learning system which handles a simple control problem on a numerical simulation of a seawater to freshwater distilling plant is given. All CPU times are on a SUN Microsystems workstation running a Motorola 68010 CPU using compiled FRANZ LISP. Appendix C contains listings of the data used and tables of results.

4.1. A Description of the Applications

Three different applications were used in testing system performance. The second of these was also used in the experiments on learning from noisy data. The freshwater distilling plant domain will be described in Section 4.5. The first application involves classifying bimetallic coordination compounds in terms of the distance between the central metal atoms. The goal is to be able to predict the metal to metal distance for new compounds. This distance is important to chemists, but is difficult to measure directly. A typical example from this domain is shown in Figure 11. A compound consists of two metal atoms, and attached to each metal atom are three to five other molecules called *ligands*. Only symmetric compounds are included in the data. The ligands on each end of the compound may be aligned with one another (eclipsed conformation) or rotated slightly (staggered conformation). Other overall characteristics of the compound, such as oxidation, covalent bond order, charge, radius of the metal atoms, and the number of electrons per metal atom are also specified. The name of each compound is also included in the data. Since ExceL cannot learn arithmetic expressions, background rules are used to partition the data into four classes according to the metal to metal distance (very–near, near, far, very–far).

The rules produced by ExceL for the chemical compound data using a confidence threshold of 0.9 and a utility threshold of 0.1 are also given in Figure 11. Note that positive exceptions have been

An example of a bimetallic coordination compound with closely spaced metal atoms.

|distance = very-near| ⇐

|NAME = "|Cr(CH3)8|4-"|	|Metal = Cr|
|Oxidation = II|	|Radius = 1.29 angstroms|
|Charge = -4|	|electrons/Metal = 12|
|Conformation = eclipsed|	|BondOrder = 4|
|Ligand1 = Me|	|Ligand2 = Me|
|Ligand3 = Me|	|Ligand4 = Me|
|Ligand5 = none|	

Rules produced by ExceL (confidence $\geq$ 0.9, utility $\geq$ 0.1)

|distance = very-near| ⇐
  |Metal = Cr v Mo||Ligand2 = R v Cl v NR3| : (1.00, 6, 6, 0) v
  |Metal = Mo||Ligand3 = Cl| : (1.00, 1, 3, 0)

|distance = near| ⇐
  |Oxidation = III||Ligand3 = X v OR v NR2 v Me| : (0.93, 14, 14, 1)
    |Metal = Mo||Ligand3 = Cl| : (1.00, 1, 1, 0)

Positive Exceptions:
  Re2Cl4|P(Et)3|4
  W2|CH2Si(CH3)3|6

|distance = far| ⇐
  |Metal = Co| : (1.00, 1, 1, 0)

|distance = very-far| ⇐
  |Ligand4 = CO| : (1.00, 6, 6, 0)

parameters: (confidence, newly covered positive examples, total covered positive examples, covered negative examples)

Figure 11. The bimetallic coordination compound domain.

enumerated using the names of the compounds. Unfortunately, these rules are unsatisfactory to chemists. A satisfactory rule would give the metal to metal distance in terms of the logarithm of the covalent bond order. Although it is not possible for ExceL (nor A^q) to find such a rule, the rules found do correctly characterise the given data.

An example of a mayfly nymph of the species *Stenonema carolina*.

|class = carolina| ⇐

|maxilla_crown_spines = 10|
|maxilla_lateral_setae = 21|
|inner_canine_teeth = 2|
|outer_canine_teeth = 7|
|terga_mid_dorsal_pale_streaks = absent|
|terga_dark_posterior_margins = absent|
|dark_marks_sterna_9 = absent|

Rules produced by ExceL (confidence $\geq$ 0.9, utility $\geq$ 0.0).

|class = carolina| ⇐
  |terga_mid_dorsal_pale_streaks = absent| : (1.00, 10, 10, 0)

|class = candidum| ⇐
  |inner_canine_teeth = 0| : (1.00, 10, 10, 0)

|class = floridense| ⇐
  |maxilla_lateral_setae = 20 .. 25||inner_canine_teeth = 4||terga_dark_posterior_margins = absent| : (1.00, 13, 13, 0)

|class = gildersleevei| ⇐
  |maxilla_crown_spines = 11 .. 13||inner_canine_teeth = 3 .. 7| : (1.00, 10, 10, 0)

|class = interpunc| ⇐
  |terga_dark_posterior_margins = present| : (1.00, 10, 10, 0)

|class = minnentonka| ⇐
  |maxilla_lateral_setae = 30 .. 40||inner_canine_teeth = 4| : (0.91, 10, 10, 1)⌊
    |terga_dark_posterior_margins = present| : (1.00, 1, 1, 0)

|class = pallidum| ⇐
  |maxilla_crown_spines = 11 .. 13||maxilla_lateral_setae = 20 .. 25| : (1.00, 10, 10, 0)

parameters: (confidence, newly covered positive examples, total covered positive examples, covered negative examples)

Figure 12. The *Stenonema* mayfly domain.

The second application involves learning rules for distinguishing seven classes of Stenonema mayflies [Lewis, 1974]. The mayflies are described in terms of a number of physical attributes such as the number spines and bristles on the upper jaw (maxilla crown spines, maxilla lateral setae), the number of teeth on

An example of the soybean disease *alternaria*.

[class = alternaria] ⇐

[canker_lesion_color = does_not_apply]
[condition_of_fruit_pods = abnormal]
[condition_of_leaves_below_affected_leaves = unaffected]
[condition_of_stem = normal]
[damaged_area = groups_upland_areas]
[external_stem_discoloration = does_not_apply]
[fruit_spots = colored_spots]
[internal_discoloration_of_stem = does_not_apply]
[leaf_malformation = absent]
[leaf_spot_color = brown]
[leaf_spot_size = greater_than_eighth_inch]
[leaf_withering_and_wilting = absent]
[margin_of_leaf_spots = water_soaked]
[plant_height = normal]
[precipitation = above_normal]
[raised_leaf_spots = absent]
[root_galls_or_cysts = does_not_apply]
[root_sclerotia = does_not_apply]
[seed_condition = abnormal]
[seed_discoloration_color = black]
[seed_shriveling = absent]
[severity = minor]
[shredding = absent]
[stem_lodging = does_not_apply]
[time_of_occurrence = October]
[color_of_spot_on_reverse_side = none]
[condition_of_leaves = abnormal]
[condition_of_roots = normal]
[cropping_history = three]
[external_decay_of_stem = does_not_apply]
[fruit_pods = diseased]
[fruiting_bodies_on_stem = does_not_apply]
[leaf_discoloration = none]
[leaf_mildew_growth = absent]
[leaf_spot_growth = with_concentric_rings]
[leaf_spots = present]
[location_of_stem_discoloration = does_not_apply]
[mycelium_on_stem = does_not_apply]
[position_of_affected_leaves = scattered_on_plant]
[premature_defoliation = present]
[reddish_canker_margin = does_not_apply]
[root_rot = does_not_apply]
[sclerotia_internal_or_external = does_not_apply]
[seed_discoloration = present]
[seed_mold_growth = absent]
[seed_size = normal]
[shot_holing = present]
[stem_cankers = does_not_apply]
[temperature = above_normal]
[yellow_leaf_spot_halos = absent]

A rule produced by ExceL for the disease *alternaria* (confidence $\geq$ 1.0, utility $\geq$ 0.0).

[class = alternaria] ⇐
    [leaf_spot_growth = with_concentric_rings v necrosis_across_veins] : (1.00, 7, 7, 0)

parameters: (confidence, newly covered positive examples, total covered positive examples, covered negative examples)

Figure 18. The soybean disease domain.

the mandibles, and the presence or absence of certain marks on the body. A typical example from the data is shown in Figure 12. Since there are no causal relationships between the descriptive attributes and the classification, the only requirements are that the rules be accurate and concise. A set of rules found by ExceL for this data set using a confidence threshold of 0.9 and a utility threshold of 0.0 (no positive

exceptions) is shown in Figure 12.

The third application involves learning the descriptions of seventeen soybean diseases. Examples of diseases are described by fifty attributes, including information about the appearance of plant stems, seeds, leaves and roots, cropping history, time of year, and distribution of damage. The data set differs from the one descibed in [Michalski and Chilauski, 1980] in that there are two more diseases, fifteen more attributes and fewer training examples included. A typical example from this domain is shown in Figure 13. The exact rule found by ExceL for the disease *alternaria* is shown in Figure 13.

4.2. Definitions of Measures Used

In all of the results shown below, a simple complexity measure is used to characterise the size of rules. The complexity of a single DNF rule is defined as the *sum of the number of complexes in the rule, plus the number of selectors in the rule, plus the number of different attributes in the rule.* The complexity of a set of rules is the average of the complexities of the rules in the set. This measure was previously used in [Reinke, 1984].

Another measure which must be defined is the error of a set of rules with respect to a set of events. The measure used here differs from those used in [Michalski and Chilauski, 1980]. Michalski and Chilauski used a syntactic distance measure and an acceptability criterion to classify an event according to a set of rules. An event was considered to be correctly classified if the correct decision was among those triggered. Since more than one decision could be triggered, the average number of different decisions triggered for testing events was represented by a separate number called the "indecision ratio." Thus, overspecialization and overgeneralisation of rules were indicated by distinct measures. A single measure is used here for both overspecialisation and overgeneralisation so that the results may compared with those of Quinlan [1983b]. For the same reason, a simple coverage test is used to classify events, although a more sophisticated evaluation scheme might be desired for obtaining lower overall error rates in a practical expert system.

In ExceL an event may be covered by several decision rules or none at all, and by one or more complexes from a decision rule. Each event belongs to one or more decision classes which for it are the *correct* (positive) decision classes, all others being *incorrect* (negative) decision classes. The error for an event

which is covered by some decision rule is defined as:

$$error(e_i) = \frac{C_N}{C_P + C_N}$$

where e_i is an event,
C_N is the number of complexes from decision rules which incorrectly cover e_i,
and C_P is the number of complexes from decision rules which correctly cover e_i.

This is the probability of making an error when randomly choosing from among the decision rules covering an event, giving stronger weight to rules for which more than one complex covers the event. For example, if there is a class "A" event which is covered by one complex from the rule for class "A" and two complexes from the rule for class "B", the error for the event is 2/3. The error for an event which is not covered by any rule is defined as:

$$error(e_i) = \frac{M - 1}{M}$$

where M is the number of decision classes.

This is the probability of being wrong when randomly assigning a decision class to an event. The percent error for a set of rules with respect to a set of events is defined as the average of the errors for the individual events:

$$Percent\ Error = \frac{\sum_{i=1}^{k} error(e_i)}{k} 100\%$$

where k is the total number of events.

4.3. Performance Comparisons

The performance of ExceL, with and without using P* as a LEF heuristic, was compared with that of AQII. AQII is an extended version of AQINTERLISP [Becker, 1983], translated to FRANZ LISP by Tony Nowicki and the author. AQII is a faithful implementation of the A^q algorithm. Like previous implementations of A^q, it does not incorporate a VL_1 data base system. The programs were run on each of the data sets described above. Table 1 gives the fundamental characteristics of these data sets.

Name	Classes	Attributes	Events
Chemistry	4	12	29
Mayfly	7	7	73
Plant	17	50	119

Table 1. Data set characteristics.

The programs were tested using the LEF's shown below:

AQII LEF = ((max-newposcovered 0.0)(min-cost 0.0)(min-selectors 0.0))

Excel LEF (a) = ((max-newpromise 0.0)(min-cost 0.0)(min-selectors 0.0))

ExceL LEF (b) = ((max-newposcovered 0.0)(min-cost 0.0)(min-selectors 0.0))

The second LEF used with ExceL is the same as the LEF used with AQII. *Max-newpromise* is the P^* measure described in Section 2.5, where P refers to only the uncovered positive events rather than all positive events. The name *promise* is used because of the close relationship between Promise and P^*. *Max-newposcovered* is a measure which simply counts the number of positive events which a complex covers, which have not been covered by some other complex in the partially completed class cover. This is typically used as the first criteria in a LEF for A^q. *Min-cost* is a measure which sums the user defined costs for all attributes in a complex. The default cost for an attribute is 1. *Min-selectors* is a measure which counts the number of selectors in a complex. *Min* and *max* indicate whether the value is to be minimized or maximised respectively. A *maxstar* value (beam search width) of 5 was used for the Mayfly data set, and a value of 4 was used for the Chemistry and Plant data sets. The *solutions* parameter of ExceL was given the same value as maxstar. The parameters to ExceL were also set to produce exact covers.

The programs were run in each of the three modes described in Section 3.3 to compare rule complexity and computation times. Rule complexity is defined in Section 4.2 above. The results for rule complexity are shown in Table 2. When forming exact covers using P^* as a LEF heuristic (case "a"), ExceL does about as well in terms of rule complexity as AQII on smaller problems, and somewhat better on larger

Data Set	Mode	AQII	ExceL (a)	ExceL (b)
Chemistry	ic	8.8	6.5	9.8
	dc	9.8	9.8	10.0
	vl	4.0	4.3	4.8
Mayfly	ic	5.0	4.7	5.6
	dc	7.3	8.0	8.0
	vl	3.6	3.6	4.4
Plant	ic	6.7	4.4	7.2
	dc	10.5	10.2	11.9
	vl	4.7	3.6	5.4

Table 2. Rule complexity comparison between AQII and ExceL with and without P^*.

problems. When the parameters to ExceL are set to allow approximate covers the rule complexity can become even lower, as will be shown in Section 4.4.3. Without using P^* as a heuristic (case "b"), ExceL produces covers which are more complicated than those produced by ExceL using P^*, or by AQII. This shows that P^* is important for finding concise descriptions when using ExceL. That AQII produces more concise covers when using the same LEF can be attributed to the fact that AQII searches a larger fraction of the search space.

Computation times were compared using the same configurations as for the rule complexity comparison. Garbage collection time was subtracted from CPU time to give a clearer indication of the computation time involved independent of the memory allocation strategy used. The data structures used in the programs are quite different, so the CPU times should only be viewed as a rough indication of the computational costs involved in each algorithm. The results are shown in Table 3.

When P^* is used as a heuristic, ExceL tends to run slightly faster than AQII on the smaller problems and much faster on the larger problems. This indicates, as expected, that computational costs for ExceL are lower than for A^q. The computation time required by ExceL without using P^* is longer than that required by ExceL using P^*. This indicates that using P^* as a LEF heuristic enables ExceL to find accept-

Data Set	Mode	AQII	ExceL (a)	ExceL (b)
Chemistry	ic	15	11	27
	dc	8	10	12
	vl	7	6	7
Mayfly	ic	22	13	29
	dc	15	13	23
	vl	17	11	22
Plant	ic	897	99	513
	dc	437	74	195
	vl	442	98	359

Table 3. CPU time comparison (in seconds) between AQII and ExceL with and without P^*.

able descriptions more quickly than previously used heuristics. Even without using P^*, ExceL is faster than AQII for larger problems.

Additional runs were made in each of the first two modes to test for differences in the predictive ability of rules produced by the two programs. The rules were generated using approximately half of the training examples in each data set. To be exact, 15 out of 29 Chemistry examples, 37 out of 73 Mayfly examples, and 68 out of 119 Plant examples were used for training, using exactly or just over half of the examples from each decision class. These rules were then tested for error against the full sets of examples. Error is defined in Section 4.2 above. Each test was repeated four times using different subsets of the training examples, and the average taken. The results are shown in Table 4.

On the average, the error rates for rules generated in all three cases are approximately equal. This should be expected since the available training information is the same in all cases. The error rates varied considerably depending on the subset of training events selected. It is likely that the differences present in the above table would be less extreme if a greater number of trials were averaged. Also, the error rates found for the Plant data are higher than those found in [Michalski and Chilauski, 1980] because: the training events were chosen randomly, not by relevant event selection program ESEL; a different error measure was used; and fewer training events were used.

Data Set	Mode	AQII	ExceL (a)	ExceL (b)
	ic	11.0	19.2	16.8
Chemistry	dc	17.3	19.8	16.0
	ic	2.6	4.0	2.3
Mayfly	dc	3.6	4.9	4.3
	ic	12.5	8.0	8.4
Plant	dc	18.8	12.4	16.8
average		11.0	11.4	10.8

Table 4. Comparison of percent rule error between AQII and ExceL with and without P^*.

4.4. The Effects of Noisy Data

An empirical study of the effects of noisy data on inductive learning was done to see how the ExceL learning algorithm performs in noisy conditions, and to replicate some of the work done by Quinlan using a version of ID3, modified to produce approximate rules, on noisy data [Quinlan, 1983b]. The *Stenonema Mayfly* data set described above was chosen for these tests because it is a real (not contrived) classification task, and the classes are well clustered (can be described by a concise rule). Also, it is small enough that the inductive learning task could be completed in a reasonable time using available resources. It differs from the data set used by Quinlan in that there are 7 equally sized classes rather than 2 classes of different sizes, and about half of the 7 attributes are redundant. That is, most subsets of 3 or 4 attributes can be used to form a correct decision rule for the given classes. These differences turn out to be important.

Noise is introduced into a data set by giving certain attributes in the training events random values. The values are selected from the domains of the corresponding attributes. Noise may be introduced into some subset of the attributes, or all of them. The *noise level* is the percentage of selectors for the chosen attributes in the data set given random values. The pseudo-random number generator used was reseeded from a real time clock to avoid repeating sequences of numbers.

4.4.1. Noise in Testing Events Only

In the first experiment, rules were generated from the original, uncorrupted training examples, then the rules were tested using a corrupted version of the data set. The parameters to ExceL were set to form exact rules. Each attribute was corrupted singly, then all attributes except the classification attribute were corrupted at once. Noise levels of 10%, 20%, 30% on up to 100% were used, and the test was repeated 10 times at each noise level. Figure 14 shows the results for this experiment.

For noise in a single attribute, a linear relationship exists between the noise level and the error rate. This agrees with Quinlan's findings for rules generated from uncorrupted data. Note that since only a single attribute is being corrupted, a particular event is either uncorrupted, or corrupted in that attribute. So, the number of corrupted events varies linearly with noise level. The error rate depends on the distribution of values for attributes in the classification rules and the number of corrupted testing events. Since the classification rules are fixed, the error rate must vary linearly with noise level.

For noise in all attributes except for the classification attribute, the error rate does *not* vary linearly with noise level. This curve can be computed from the data found for single attribute noise using the *principle of inclusion and exclusion* [Liu, 1977]:

$$|A_1 \bigcup A_2 \bigcup \cdots \bigcup A_r| = \sum_i |A_i| - \sum_{1 \le i < j \le r} |A_i \bigcap A_j| + \sum_{1 \le i < j < k \le r} |A_i \bigcap A_j \bigcap A_k|$$
$$+ \ldots + \left(-1\right)^{r-1} |A_1 \bigcap A_2 \bigcap \cdots \bigcap A_r|$$

where A_i is a set of objects.

For the current problem, A_i is a set of events which are classified incorrectly due to noise in the i-th attribute. Two simplifying assumptions are needed to apply this formula. First, it must be assumed that an event is either classified correctly (error = 0), or classified incorrectly (error = 1). Second, it must be assumed that an event which has several noisy attributes, any one of which would independently cause the event to have an error of 1, still has an error of 1 (i.e. an event can only count as one error, and two wrongs don't make a right). The available information for single attribute noise gives the *percentage* of all events which are in error due to noise in a single attribute. The cardinality of intersecting sets of

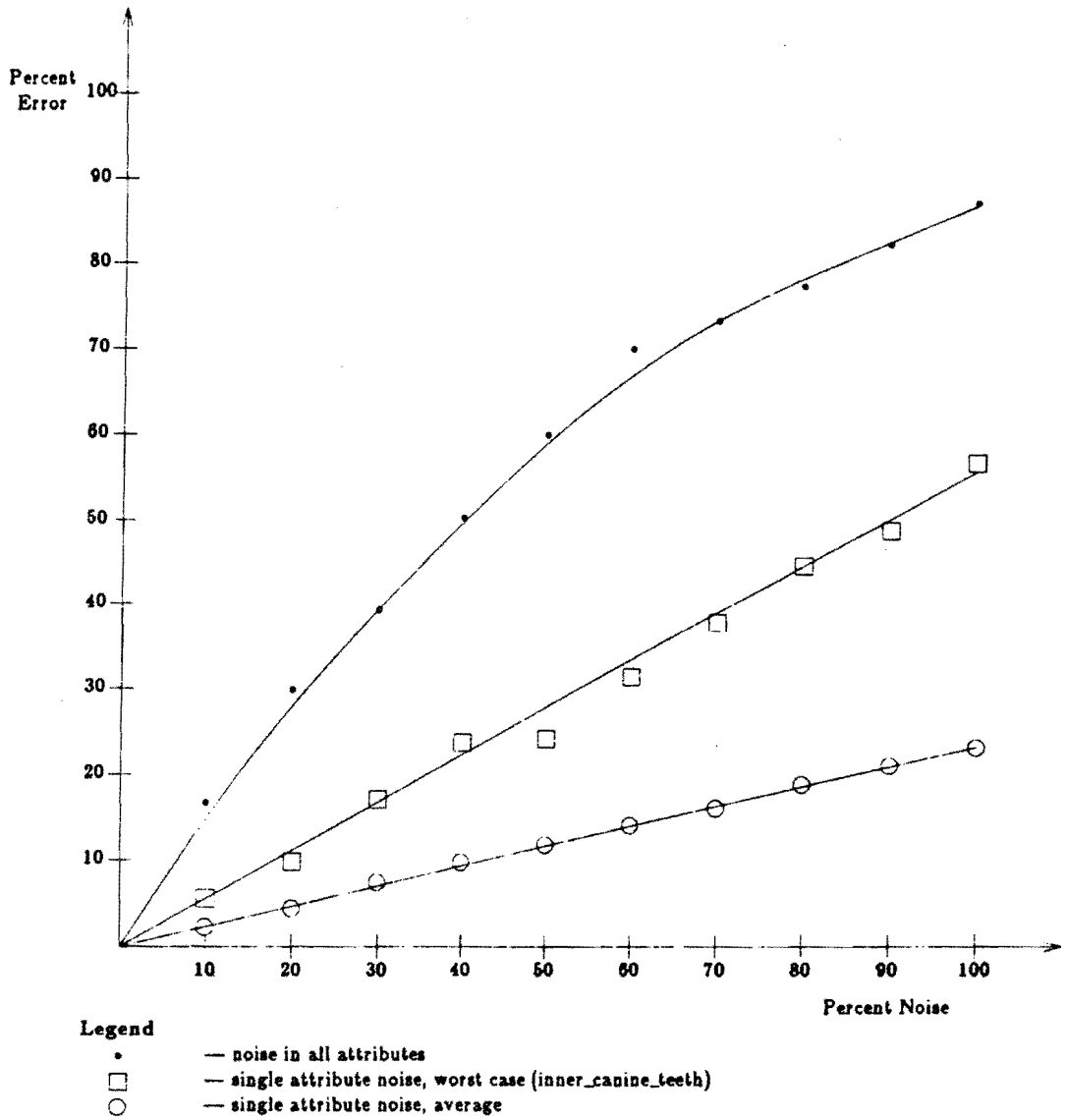

Figure 14. Classification error with noise in testing events only.

incorrectly classified events can be computed by simply multiplying these percentages (as fractions) since the distribution of events is random. Computing the error for noise in all attributes by combining the errors for single attribute noise in this way gives values almost identical to those found empirically (see Appendix C.3). Since the principle of inclusion and exclusion can be applied to combine any subset of

noisy attributes, it can be concluded that a non-linear relationship between error and noise will be observed any time more than one attribute which appears in the classification rules is noisy.

4.4.2. Noise in Both Training and Testing Examples

In the second experiment, the data set was corrupted to a certain noise level, a set of rules was generated using the noisy data, then the rules were tested using a different randomly corrupted data set. All examples in the data set were used. The parameters to ExceL were set to form exact rules. Conflicting events were dropped from the data set. Each attribute was corrupted singly, then all attributes except the classification attribute were corrupted at once, then the classification attribute was corrupted. Noise levels of 10%, 20%, 30% on up to 100% were used, and the test was repeated 5 times at each noise level. Figure 15 shows the results for the experiment with noise in all attributes, with noise in the classification attribute, with noise in a single attribute (highest error), and with noise in a single, less important attribute.

For single attribute noise, the error rates are much lower when rules are generated from noisy data than when they are generated from uncorrupted data. The shape of the resulting curve depends on the importance of the attribute. For the most important attribute (maxilla_lateral_setae) there is a saturation effect of sorts. For less important attributes (such as inner_canine_teeth) there is a noticeable drop–off in error at higher noise levels. This can be attributed to redundancy in the data. At higher noise levels, a noisy, less important attribute is not useful for discrimination between the classes and is not used in the rules. Since there is sufficient redundancy in the data, ignoring the noisy attribute leads to lower error rates. Quinlan did not notice the drop–off effect, probably because the data set he used contained only 1 redundant attribute out of 39. Similarly, more important attributes are used less frequently in the rules as the noise level increases so the error rate is lower than for rules produced from uncorrupted data.

One important difference from Quinlan's results is that the attribute with the highest error rate for single attribute noise in the first experiment is *different* from the one in the second experiment, but in Quinlan's tests they were the *same*. This is due to the way the learning algorithm used here generates rules. Rules are formed by choosing one class to be the positive class, and making all others negative. When this is done, an attribute which is important overall for a multi–class discrimination often becomes

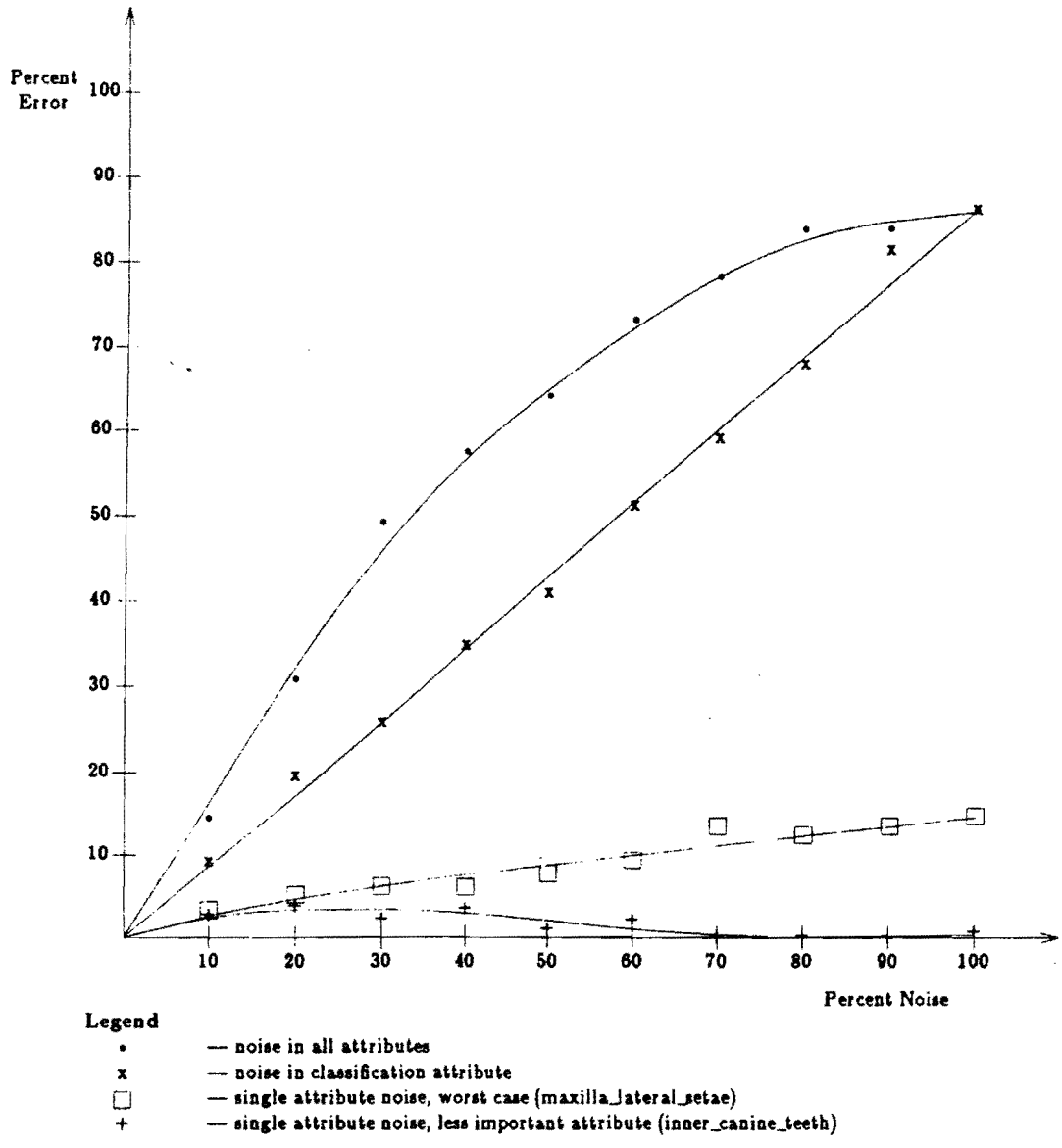

Figure 15. Classification error with noise in training and testing events.

less important for a particular two–class discrimination.

For noise in all attributes except the classification attribute, the saturation effect is exactly as found by Quinlan. The peak rate of error at about 86% corresponds to the probability of guessing wrong in a 7

class decision problem. There are two sources of error when noise is present both during training and testing: (1) error due to uncorrupted events being misclassified by rules generated from noisy data, and (2) error due to noise in the testing events. These errors are compounded at lower noise levels, but the effect of (1) diminishes at higher noise levels since there are fewer uncorrupted events. The result is the saturation effect seen in the graph. Comparing this to the graph for the case of noise in all attributes when noise is present only in the testing events (Figure 14), the rate of error is found to be somewhat higher, as should be expected.

For noise in the classification attribute, there is a linear relationship between noise and error. The linearity is partly due to a trick in the way that error is defined and tested for. Note that to test for error in assigning an event to a class, the correct classification must be known. Thus, classification error in the *testing* events is essentially ignored – that is why this case does not appear in the previous experiment, the error would always by 0%. Since noise occurs only in one attribute, the classification attribute, the percentage of all events which are corrupted will be the same as the noise level. The probability of an event being placed in the wrong class is:

$$\frac{M-1}{M}$$

where M is the number of decision classes.

So the error for noise in the classification attribute is:

$$Percent\ Error = \eta \frac{M-1}{M}$$

where η is the noise level in percents. This explains why the graph for noise in the classification attribute is linear.

4.4.3. Noise and Approximate Decision Rules

An approximate decision rule is one which does not cover all of the observed events in an event space correctly. This can occur by allowing exceptions to the rules or by training on a subset of the available data. The ExceL algorithm is capable of forming approximate decision rules by allowing positive and negative exceptions. The degree to which positive exceptions are allowed is controlled by specifying a

minimum *utility* that a complex in a rule must have. The degree to which negative exceptions are allowed is controlled by the *confidence* that a complex in a rule must have. Exact rules are formed by using a confidence of 1.0 and a utility of 0.0. The experiment with classification error in both training and testing events was repeated to determine how classification error, rule complexity, and CPU time required for rule formation are affected when the decision rules are allowed to be approximate. The test was run with (A) a confidence of 1.0 and a utility of 0.0, (B) a confidence of 0.9 and a utility of 0.1, and (C) a confidence of 0.8 and a utility of 0.2.

The *shape* of all graphs for error with respect to noise are the same as in Figure 15. In addition, the levels of error changed little when the decision rules were allowed to be approximate. The most interesting figures are given below. Maximum noise levels were determined from the average of the three values near the peak.

Error Figures for Approximate Decision Rules

Data Point	Run A	Run B	Run C
Maximum of Single Attribute Noise (maxilla_lateral_setae)	13.6%	11.4%	10.0%
Maximum of Single Attribute Noise (inner_canine_teeth)	3.0%	4.4%	4.9%
Maximum of Average Single Attribute Noise	2.5%	3.4%	3.5%

For single attribute noise, the error rates increased modestly on the average. The error rates for less important attributes tended to increase, while the error rate for the most important attribute (maxilla_lateral_setae) tended to decrease. There is a noticeable drop–off in error rates at higher noise levels for single attribute noise in most of the less important attributes in Run B and Run C. For the case of noise in all attributes, there seemed to be a slight trend towards higher error rates at noise levels below 40%, but the trend is not strong enough to draw conclusions from. The curves peaked at about 86% in all runs. For the case of noise in the classification attribute, the curves are identical for all levels of precision.

If the rate of error is not changed significantly, what is gained by using approximate rules? Figure 16 shows the change in rule complexity with respect to noise for the three levels of precision tried, for the case of noise in all attributes, and Figure 17 shows the change in CPU time required. Noise in class membership information produced a similar but less dramatic effect, and single attribute noise had only a

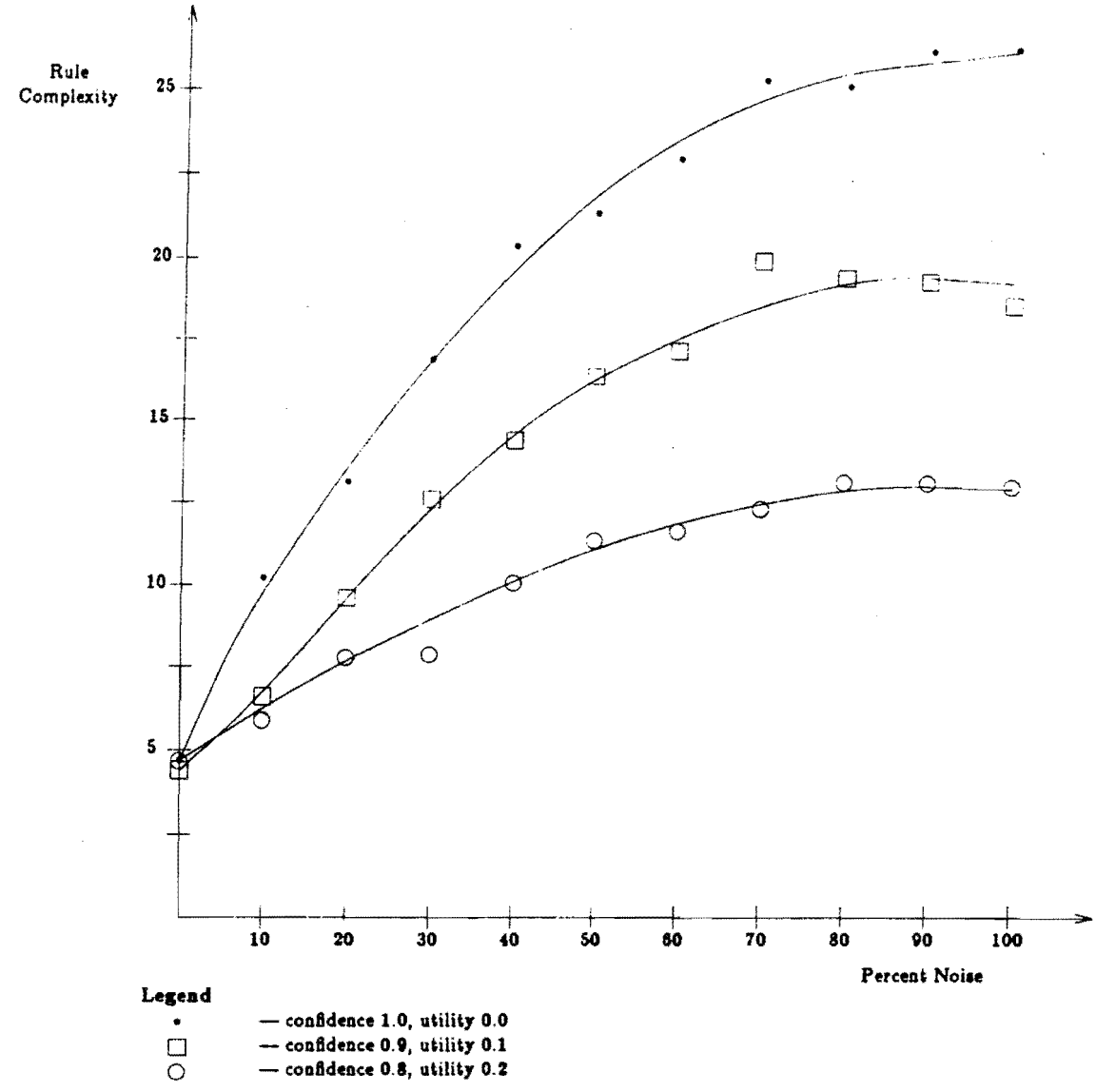

Figure 16. Rule complexity with varying precision, noise in all attributes.

small effect on rule complexity and CPU time. For high levels of noise, especially noise spread across all attributes and noise in class information, approximate decision rules are desirable because they are simpler and can be generated more quickly. The fact that the error rate is changed little indicates that the simplifications made by ExceL are appropriate. The algorithm makes good choices about which events

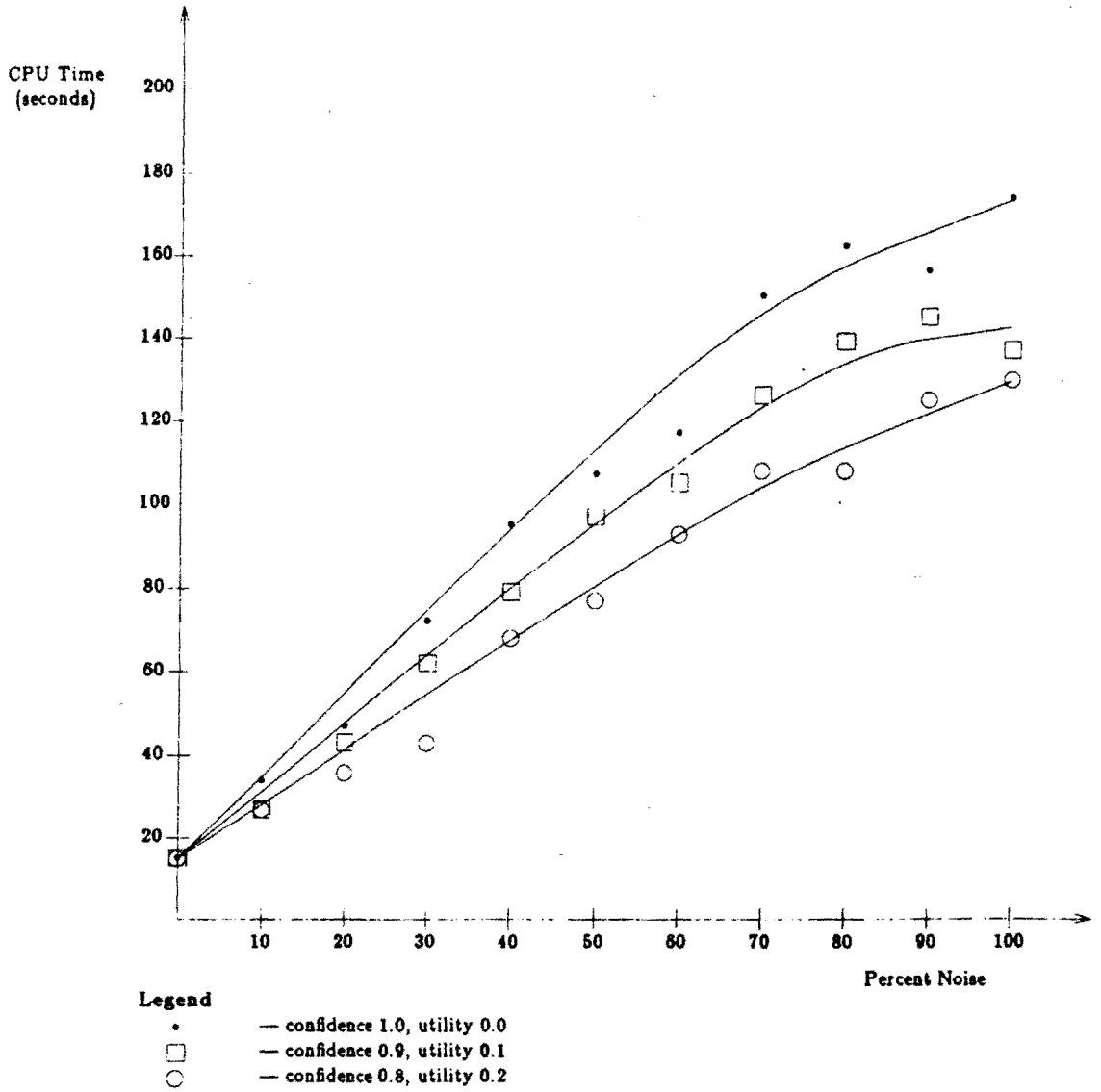

Figure 17. CPU time with varying rule precision, noise in all attributes.

should become exceptions.

4.4.4. Discussion of Quinlan's Findings

Quinlan reports a number of "policy-level implications for the use of inductive inference machinery in noisy environments" based on the results of his experiments [Quinlan, 1983b]:

(1) It is important to eliminate noise regarding class membership of the objects in the training set.
(2) It is not worthwhile expending effort to eliminate noise from the attribute values of objects in the training set if there is going to be a significant amount of noise when the resulting classification is used in practice.
(3) We are better off dispensing altogether with noisy, less important attributes.
(4) The payoff in noise reduction increases with the importance of the attribute.
(5) Noise is less of a problem with approximate classification rules.

Each of these points will be addressed in turn.

(1) Noise in *all* attributes at once produces rates of error higher than noise in class membership information at all noise levels. In addition to avoiding noise regarding class membership of the objects in the training set, when there is noise in all attributes it is best to use *uncorrupted* data for generating rules if possible. This view can be supported by comparing the curves for noise in all attributes in the Figures 14 and 15 above.

(2) This recommendation is supported by these new findings. In addition, if redundant attributes exist and the noise tends to be restricted to just a few attributes, rules generated from noisy data *will perform much better* than those generated from correct data. This is because the learning algorithm tends to avoid using the noisy attributes in the rules generated.

(3) Noisy, less important attributes do introduce a small amount of error. The peek error rate for noisy, less important attributes tends to occur at noise levels between 20% and 40%. Above that, the attributes are not used as frequently in the rules generated and thus do not contribute as much to error. Noisy, less important attributes are probably more of a problem when the data does not contain many redundant attributes.

(4) The payoff in noise reduction would increase with the importance of the attribute. The more important attributes had the highest rates of error when corrupted singly.

(5) Noise is not really a problem for *exact* decision rules, except that the rules are much more complex than approximate decision rules. The only noise handling technique used when forming exact decisions was dropping conflicting events if they happened to be generated by the introduction of noise (this happened rarely). If noise is present, it is desirable to produce approximate decision rules because the rules will be less complex, and the rules can be generated more quickly.

Overall, the most important finding seems to be that if the data is noisy it is desirable to have redundancy. There must be *some* correct information for the system to work with else it cannot generate rules with any predictive power. This is not at all remarkable from the point of view of information theory. The ExceL system is, as probably most related inductive learning systems are, capable of ignoring isolated noisy attributes as long as there are enough uncorrupted attributes to work with.

The ExceL and ID3 inductive learning algorithms both learn rules for describing classes of objects, but they do so in different ways. It is significant that different inductive learning systems which generate rules in different forms working on different data sets should yield such similar results. The mathematical basis for two effects has been given, and the validity of other phenomena noted by Quinlan has been supported.

4.6. A Closed Loop Learning System

The final experiment is a demonstration of closed loop learning system capabilities on a simple control problem. The application involves learning situation–action rules for controlling five valves in the seawater system of a seawater to freshwater distilling plant (evaporator) [Machinist Mate 3 & 2, 1972]. These valves are manually controlled on older models of these devices, and considerable operator attention is required to keep the levels in the three main containers within the correct operating range. A simple mechanical level controller can be used to reduce the need for human intervention so this work is not being promoted as a practical solution to this particular problem. Rather it is a demonstration of learning system capabilities in the general area of control problems. Data derived from the simulator is used to examine performance improvement as the number of observed training events increases (i.e. the *learning curve*) for rules of varying precision.

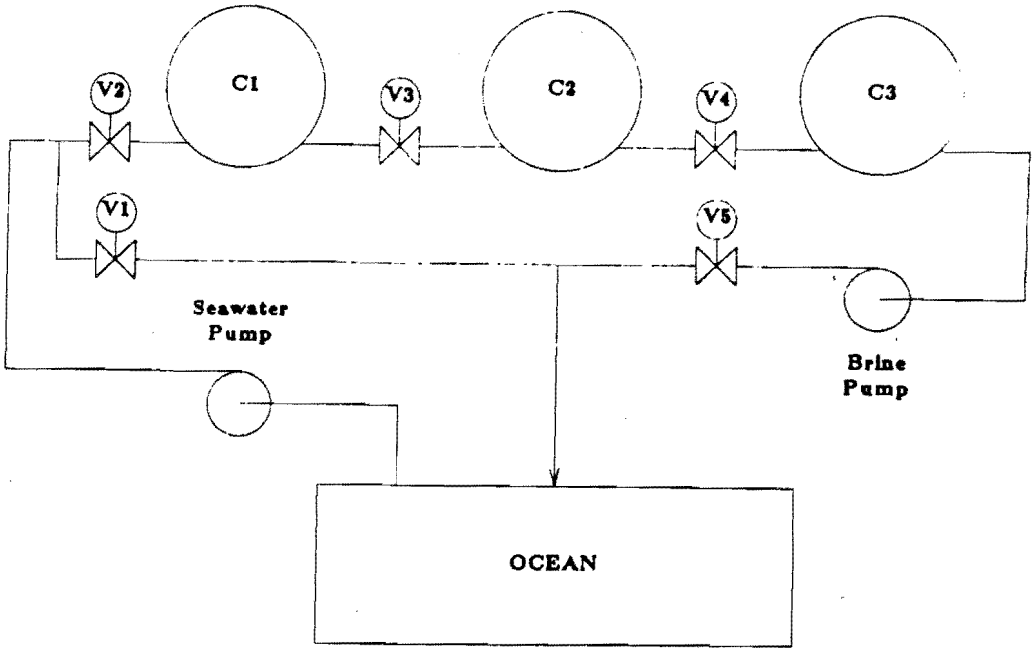

Figure 18. Triple effect evaporator seawater system.

A numerical simulation of the system shown in Figure 18 is used to interact with the rule interpreter and rule learning systems. The system consists of two pumps, one for supplying seawater to the evaporator, and one for removing brine. The three containers labelled C1, C2 and C3 are low pressure, low temperature boilers. C3 operates at the highest vacuum and C1 operates at the lowest vacuum. The pressure differential moves the seawater between the containers. The five valves are used to maintain the correct levels in the containers, which for this simulation is set at between 450 and 550 gallons. The values used in the system for flows, capacities, etc. are fictitious, but the overall behavior of the system is qualitatively correct. Background rules are used to convert the numerical values for container levels and changes in level into qualitative values.

A training example for this application is shown in Figure 19. The selectors centered at the top give the correct actions for each of the five valves in the situation indicated by the conjunction of selectors

A training example for freshwater distilling plant control.

|adjust-v1 = open| |adjust-v2 = shut| |adjust-v3 = shut| |adjust-v4 = open| |adjust-v5 = ok| ⇐

|pump(p1) = on|
|pump(p2) = on|
|position(v1) = 0.6|
|position(v2) = 0.2|
|position(v3) = 0.8|
|position(v4) = 0.5|
|position(v5) = 0.2|
|C1-level = high|
|C2-level = ok|
|C3-level = high|
|C1-change = up|
|C2-change = up|
|C3-change = down|

Final rules produced by ExceL for valve v2 (confidence ≥ 1.0, utility ≥ 0.0).

|adjust-v2 = open| ⇐
    |C1-change = down .. steady||C1-level = low| : (1.00, 32, 32, 0) ∨
    |C1-change = down||C1-level = ok| : (1.00, 15, 15, 0)

|adjust-v2 = ok| ⇐
    |C1-change = up||C1-level = low| : (1.00, 26, 26, 0) ∨
    |C1-change = steady||C1-level = ok| : (1.00, 35, 35, 0) ∨
    |C1-change = down||C1-level = high| : (1.00, 19, 19, 0)

|adjust-v2 = shut| ⇐
    |C1-change = steady .. up||C1-level = high| : (1.00, 27, 27, 0) ∨
    |C1-change = up||C1-level = ok| : (1.00, 19, 19, 0)

parameters: (confidence, newly covered positive examples, total covered positive examples, covered negative examples)

Figure 19. The freshwater distilling plant control domain.

below. The information given is the states of the two pumps (on or off), the position of each valve (0.0 is fully closed, 1.0 is fully open), the level in each container (high, ok, or low), and the change in container level between the last observed level and the current level (up, steady, or down).

There are fifteen rules which the system learns, three for each valve (open, ok, shut). An open or shut action moves the valve by one-tenth of its range in the appropriate direction. The final rules learned by the system for controlling the valve labelled v2 are also shown in Figure 19. The rules for controlling valves v3 and v4 are similar. The rules for controlling valves v1 and v5 move the valve to a fixed position, as is usually done in actual operation.

The rule interpretation cycle shown in Figure 5 (Section 2.4) is used for this application. The *sensor* is a program module which reads the state of the freshwater distilling plant simulator. The *effector* is another program module which modifies the positions of the simulator valves. No internal state is needed, so the internal state is always given as *all*. The dialog given below shows several cycles from an interactive training session. This demonstrates performance early on in the learning process – about forty training events have been observed so far. Input typed by the critic is shown in bold face, comments are in italics:

First, the program prints out the state of the simulator for the critic. The level in container C1 is low and steady, the level in container C2 is high and steady, and the level in container C3 is low and steady.

External State:
[pump(p1) = on][pump(p2) = on]
[pos(v1) = 0.3][pos(v2) = 0.2][pos(v3) = 0.7][pos(v4) = 0.1][pos(v5) = 0.6]
[C1–change = steady][C1–level = low]
[C2–change = steady][C2–level = high]
[C3–change = steady][C3–level = low]

Internal State:
all

The system advises opening valve v1, shutting valves v3 and v5, and leaving valve v4 as is. Conflicting rules are triggered for valve v2, giving the "?". set of actions fired by the rule interpreter.

Advice:
[adjust–v1 = open][adjust–v2 = ?][adjust–v3 = shut][adjust–v4 = ok][adjust–v5 = shut]

The correct set of actions according to the critic is to open valves v1, v2, and v4, and to shut v3 and v5.

action ?> open v1
action ?> open v2
action ?> shut v3
action ?> open v4
action ?> shut v5
action ?> ok

The system and critic disagree on actions for valves v2 and v4 so the rules must be modified. Training examples are generated by combining the correct action as given by the critic for valves v2 and v4 with simulator state as given above. The learning system then specializes the conditions for leaving valves v2 and v4 as is, and generalizes the condition for opening valve v4.

Spec: [adjust–v4 = ok]
Spec: [adjust–v2 = ok]
Gen: [adjust–v4 = open]

At this point, the "effector" is applied to adjust the valve positions, and the simulator is run for an interval of time to allow flows and levels to change. Again, the interpreter state (obtained from the "sensor") is printed out for the critic. The level in container C1 is still low and steady. The level in container C2 is still high and now is rising. The level in container C3 is still low and now is dropping.

External State:
[pump(p1) = on][pump(p2) = on]
[pos(v1) = 0.4][pos(v2) = 0.3][pos(v3) = 0.6][pos(v4) = 0.2][pos(v5) = 0.5]
[C1-change = steady][C1-level = low]
[C2-change = up][C2-level = high]
[C3-change = down][C3-level = low]

Internal State:
all

The system advises opening valves v1, v2 and v4 and shutting valves v3 and v5.

Advice:
[adjust-v1 = open][adjust-v2 = open][adjust-v3 = shut][adjust-v4 = open][adjust-v5 = shut]

The critic agrees, so no training examples are generated and no changes are made to the rules. Note that the action recommended by the system when the level in container C1 is low and steady is now correct.

action ?> **open v1**
action ?> **open v2**
action ?> **shut v3**
action ?> **open v4**
action ?> **shut v5**
action ?> **ok**

The simulator valves are adjusted, the simulator is run for a time, and the interpreter state is printed out. The level in container C1 is low and rising. The level in container C2 is still high and rising. The level in container C3 is still low and dropping.

External State:
[pump(p1) = on][pump(p2) = on]
[pos(v1) = 0.5][pos(v2) = 0.4][pos(v3) = 0.5][pos(v4) = 0.3][pos(v5) = 0.4]
[C1-change = up][C1-level = low]
[C2-change = up][C2-level = high]
[C3-change = down][C3-level = low]

Internal State:
all

The system recommends leaving valve v1 as is, shutting valves v3 and v5, and opening valve v4. No rule is triggered for valve v2.

Advice:
[adjust-v1 = ok][adjust-v3 = shut][adjust-v4 = open][adjust-v5 = shut]

The critic agrees except for indicating that valve v2 should be left as is. A training example is generated for the using the action given by the critic for valve v2 and the simulator state given above.

```
action ?> shut v3
action ?> open v4
action ?> shut v5
action ?> ok
```

The learning system then generalizes the condition for leaving valve v2 as is.

Gen: [adjust–v2 = ok]

It was found that the system does not learn complete rules by observing the critic if the simulator is always kept near the correct operating zone. This is to be expected since a learning system cannot be expected to learn something which it has not been taught. The significance of this observation is that it would be difficult to train a control system using this technique if complete training involved placing the device being controlled in an unsafe condition. For example, it would not be desirable to teach water level control of high temperature boilers if allowing the water level to become very low could lead to melting the boiler. Some other form of training might be needed in this case, such as learning characteristic descriptions from examples within the safe range and learning by being told or by using fabricated examples outside of the safe range. In addition, it was found that large numbers of training examples were needed so that the learning system would avoid using irrelevant attributes which by chance had regular patterns of values in small sets of examples.

About 180 training examples were sufficient for the learning system to generate good rules containing only a few irrelevant variables. Figure 20 shows how the system performed as a freshwater distilling plant operator after an extended training period. The critic function was modified to always agree with the advice given by the production system rather than query a human expert, and the simulator was started in an unstable condition well out of the normal operating range. All levels were brought within the operating range quickly. Some oscillation occured because only coarse adjustments of valve position could be made, but the levels usually stayed within the operating range once stabilised.

The large set of training examples accumulated for this domain was used to examine performance improvement as the number of observed training examples increases for varying levels of rule precision

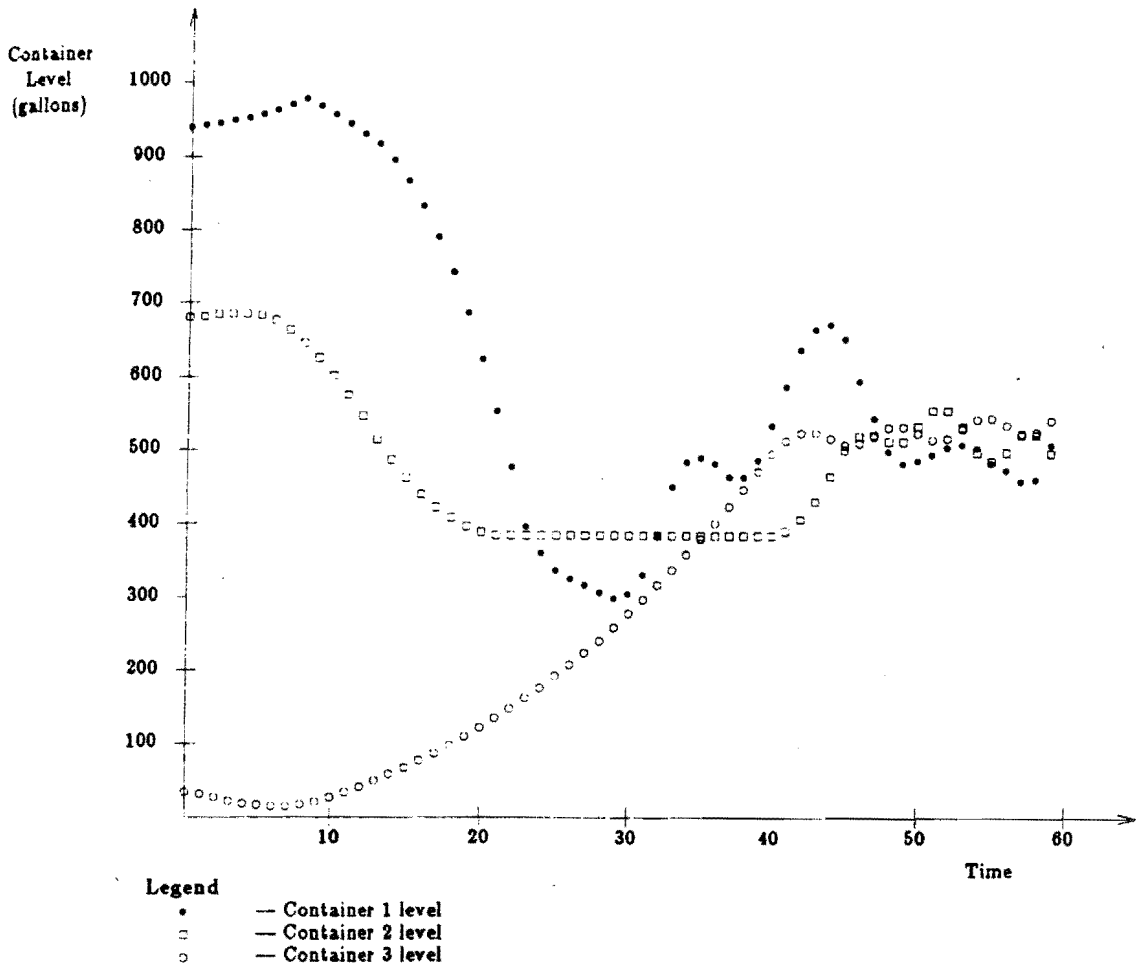

Figure 20. Trace of freshwater distilling plant container levels under automatic control.

Rules were generated using from 3% to 100% of the observed examples (the data point shown for 0% of the observed examples is the error rate for random guessing). Rules were generated at three different levels of precision by setting the confidence and utility parameters at 0.8 and 0.2, 0.9 and 0.1, and 1.0 and 0.0 respectively. The rules generated were then tested against the set of all available training examples.

Figure 21 shows the results of this experiment. There is little difference between the error rates for approximate and exact rules when less than 10% of the available training examples are used. The

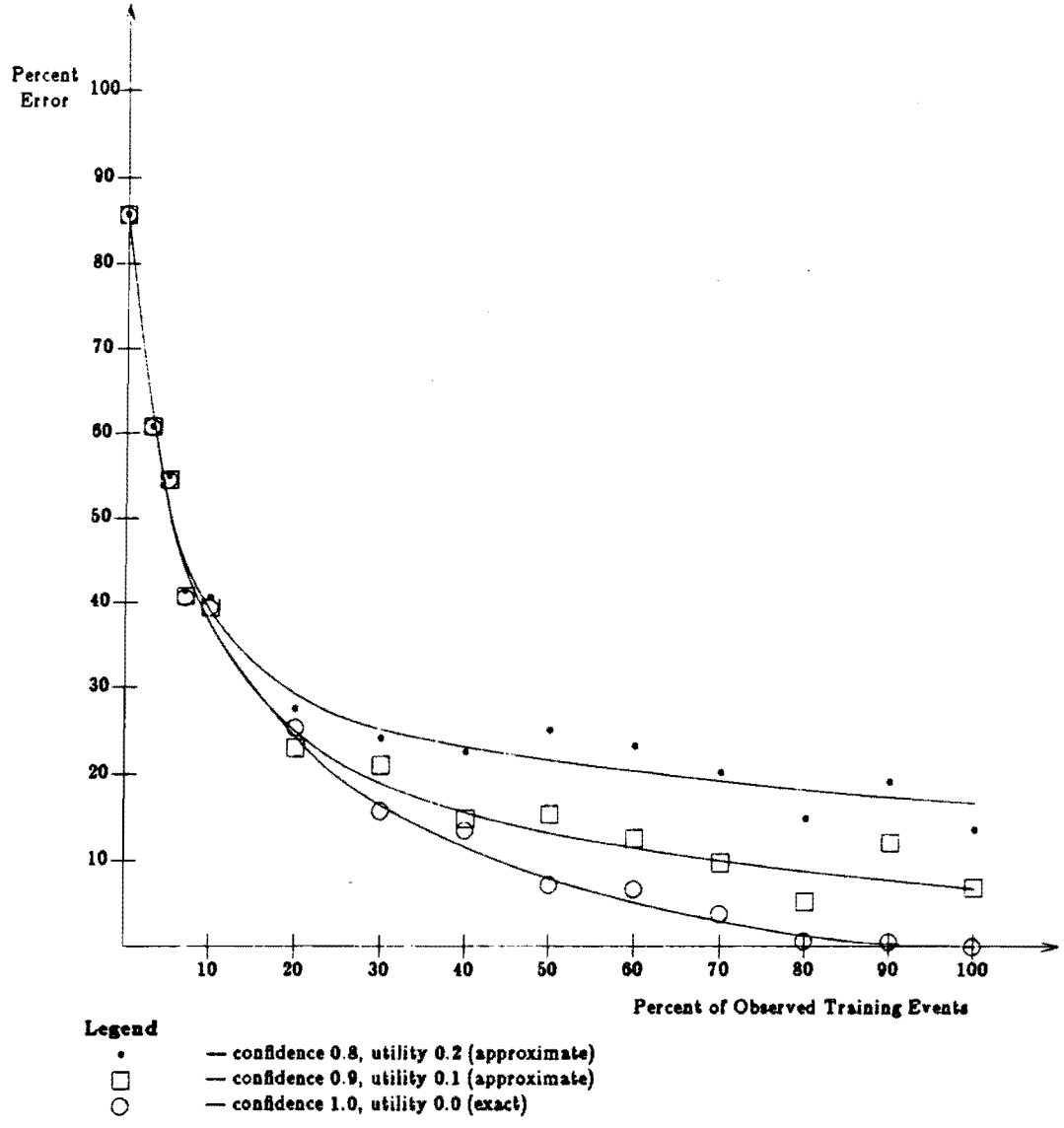

Figure 21. Learning curves for rules of varying precision.

difference increases as the number of observed examples increases. The error rate for all rules converges to a level corresponding to the confidence threshold used in generating the rules. Note that since both training and testing examples are uncorrupted, we have

$$\text{Error} = 1 - \text{Confidence}$$

when all available examples are used for training. This should be expected because the definition of *confidence* used here is based on the relative frequency of occurence of correctly covered events (Section 2.3.2).

This demonstrates that rules of a specified degree of precision can be learned. With negative exceptions summarized using *unless* conditions, and using an appropriate interpreter, deductive inference could also be done with a specified degree of precision.

5. CONCLUSIONS

A system for learning decision rules with exceptions has been described and its capabilities evaluated. The system is capable of learning from noisy and conflicting data, and can generate *unless* conditions to summarise legitimate exceptions. A closed loop learning system which can apply these rules has been used to demonstrate learning capabilities in the area of control problems, and to study performance improvement over time with varying rule precision.

5.1. System Performance

The effects of learning from noisy data have been examined using ExceL as a replication of experiments performed by Quinlan using a modified version of ID3 [Quinlan, 1983b]. Some of the results were shown to have simple mathematical explanations. Several observations were made concerning redundancy and noisy data. In particular, when noise is present, redundancy in the training data is desirable. If redundancy is available and noise is limited to a few attributes in both the testing and training data, then it is best to train on noisy data. And, if noise is distributed evenly across all attributes, it is best to train on uncorrupted data.

The effect of noisy data when using approximate decision rules was also examined. It was found that when the data is noisy, approximate decision rules are almost as accurate as exact decision rules, even though the complexity of the approximate rules is much less. These tests also indicate that ExceL does a good job of isolating noisy events as exceptions when forming approximate rules. It would be interesting to see if noise immunity would be improved by classifying an event according to a set of rules using a measure which indicates the degree of match (such as the *degree of consonance* [Michalski and Larson, 1983]) rather than a boolean coverage test.

For the data sets tested, the ExceL system usually generated more concise covers than A^q, and did so in less time. No significant difference in the accuracy of the rules was observed. The improvement in performance has been attributed to a better description evaluation heuristic, focusing on inconsistencies, and implementation details such as the use of a data base system and caching frequently used information.

The need for preprocessors such as the VARSEL program for relevant attribute selection [Baim, 1984] and the ESEL program for relevant event selection [Cramm, 1983] as efficiency boosters is greatly reduced. In effect, many of the principles of these programs are incorporated into the ExceL algorithm. When a preprocessor is used there is a chance that valuable information will be lost and rule quality degraded, so it is desirable for a learning algorithm to be able to use all available information efficiently.

A closed loop learning system has been used to demonstrate the learning capabilities of the system, and to examine performance improvement over time with varying levels of rule precision. It was shown that the learned rules could in fact be used to control the freshwater distilling plant simulator. But, it was found that the process of interactive learning using this closed loop system requires operating the simulator outside of normal limits. This approach would have to be modified for devices which are unsafe when operated outside of normal limits. Also, a large number of training examples were needed before the learning system could determine which attributes were relevant. In the area of performance improvement, it was found that the difference in error rate between exact and less precise rules increases as the percentage of observed events increases. The error rate converges to a level corresponding to the confidence threshold used in generating the rules, so rule of a predetermined level of precision can be generated.

5.2. Limitations and Future Directions

In ExceL, the degree to which exceptions are allowed in inductively learned rules is controlled by user defined parameters for rule confidence and utility. This is much like specifying confidence thresholds for triggering rules in a deduction system. It would be desirable to eliminate the need for specifying these thresholds, possibly by allowing exceptions only when this makes a rule simpler according to a complexity criterion.

Current inductive learning systems are very good at recognizing patterns in training data. However, the operators available for combining features in current systems are sometimes not adequate. For example, the rules that ExceL generates for the *chemistry* data set described in this paper are worthless to an experienced chemist. The main problem is that concomitant variation of attributes [Copi, 1982] is not noticed by the system. When a decision is based on a linear attribute such as distance, the system has no

way to recognise nor to represent proportionalities or mathematical relations between the decision attribute and the descriptive attributes. A possible solution to this problem is to combine the current system with a system designed specifically for finding *mathematical* relations in data when a decision is based on a linear attribute such as Abacus [Falkenhainer, 1984].

Another problem is that a simple rule which accurately describes the available data may not make sense as far as a human expert is concerned, even though the system has the representational power needed. This usually occurs because the system has no information about the problem domain which could be used to guide induction, especially information about causal relations between the descriptive attributes and the classification attributes. This is a problem with empirical learning systems in general, not just the system described in this thesis. A stopgap measure to handle this problem is to allow the user to advise the system by assigning cost and relevance weights to attributes. A better solution may be to provide the system with information about causal relations and other domain specific information (preferably by a learning process) and allow it to do reasoning about the relevance of attributes for making particular decisions from this information. For example, in the freshwater distilling plant problem, a large number of examples must be accumulated before the learning system recognizes that levels and changes in level of immediately adjacent containers are the most relevant attributes for the decision of how to adjust a particular valve. Knowledge about piping systems and fluid flow might simplify this learning task.

The incorporation of domain specific knowledge in machine learning has been investigated by Michalski [1980; 1983]. Michalski advocates using well defined knowledge representation languages, general purpose induction algorithms, and modularity when including domain specific knowledge. Research in the area of machine learning using domain specific knowledge to guide induction is also being conducted by DeJong [1983], and Mitchell [Mitchell, Mahadevan, and Steinberg, 1985], among others. The knowledge representation schemes used by these researchers tend to be ad hoc since their approach is to find a powerful solution to a particular problem rather than to find a general purpose but perhaps less powerful solution for a wide class of problems. The ExceL system could be extended to incorporate additional background knowledge by defining new types of background knowledge rules and corresponding inference

mechanisms. Developing general purpose techniques for incorporating domain specific knowledge in induction programs is an ongoing research task.

Incremental learning tends to be difficult in practice because of the bookkeeping involved. It is necessary to keep track of the examples associated with each decision rule, and the rules are continuously being modified. All of the most difficult problems of data base manipulation are involved. The data base system currently used in ExceL is fine for short term rule development tasks, but a more sophisticated data base system which provides a convenient mechanism for long term storage and is easy to interface to is needed. It should be possible to extend the system used in ExceL to fulfill this need at least in part.

Certain aspects of incremental learning were not dealt with. For example, it would be desirable to be able to introduce new decision classes during the learning process. Also, it should be possible to use previously learned concepts (such as knowledge of fluid flow) in the process of learning new concepts so that the learning task can be partitioned. The first shortcoming can be handled by the generalization/specialization mechanism currently used. The second shortcoming could be corrected to some degree using the background rule mechanism.

The use of VL_1 as a representation language is in itself a limitation. Relationship information must be represented in attribute form rather than directly in structural form. For example, the predicate

adjacent (A, B)

must be represented in attribute form as

[adjacent–A–B = true].

This is also a drawback when attempting to make rules about such things as opening and closing of valves in general rather than opening and closing *particular* valves. Extensions are necessary to represent interesting background rules and to provide an interesting rule interpreter. Learning systems have been developed based on more powerful representation languages. INDUCE–2 learns rules in VL_{21}, a many–sorted predicate logic language [Hoff, Michalski, and Stepp, 1983], and Marvin learns rules in PROLOG [Sammut, 1981; Sammut and Banerji, 1983]. Solving learning problems using INDUCE–2 is computationally very costly, and Marvin requires training examples that are carefully sequenced in order to achieve

any results. Research should continue in the area of machine learning using more powerful representation languages such as Annotated Predicate Calculus [Michalski, 1983].

APPENDIX A: GLOSSARY

There are a large number of words used in unique ways in the methodology described in this thesis. This appendix provides an introduction to this terminology. For a more formal description of the variable valued logic system VL_1 see [Michalski, 1974].

Attribute

An *attribute*, also frequently called a *variable*, is some aspect of an object to be described such as color, temperature, shape, weight, etc. Associated with each attribute is a set of values which the attribute may assume called its *domain*. There are four fundamental domain types in use:

- A *nominal* domain is a set of unordered discrete values. For example, the attribute *color* may have values *red, green, orange, etc.*
- A *linear* domain is a set of ordered discrete values. For example, the attribute *temperature* may have values *32 .. 105 degrees.* The ".." is the "range operator" and is used to specify a segment of values as shown here.
- A *cyclic* domain is a set of ordered discrete values, where the first element is the successor of the last element. For example, the attribute *day* may have values *Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday,* where Monday follows Sunday.
- A *structured* domain is a set of unordered discrete values in a tree structure, where the tree structure represents an abstraction hierarchy of descriptions. For example, *pentagon, hexagon, and octagon* may all be siblings of the node *polygon* for the attribute *shape.*

Selector

A *selector* consists of an attribute, a relational operator, and a single value or a disjunctive list of values from the domain of the attribute. The relation must be one of:

$$= \quad \neq \quad > \quad < \quad \geq \quad \leq$$

A selector is a predicate which is either satisfied (true) or not satisfied (false) in a particular context. The selector [color = red] would be read "the color is red" for the particular object in question. The selector

[shape = round v pointed] would be read "the shape is round or pointed." This is an example of "internal disjunction."

Event Space

An *event space* is the space of all possible descriptions as defined by a given set of attributes and their corresponding domains. That is, it is the cartesian product of the domain sets. This is essentially the same concept as a "feature space" used in pattern recogition, except that a feature space is typically described only by linear domain attributes while an event space may be described by any combination of nominal, linear, cyclic. or structured domain attributes.

Complex

A *complex*, formally speaking, is a cover of a portion of an event space. In this thesis, a complex is usually a conjunction of selectors. For example, the complex

$$[\text{color} = \text{red}][\text{shape} = \text{round v pointed}]$$

is a description for all objects which are red and have either a round or pointed shape.

Event

An *event* is a complex which describes a single point in an event space. An event has one, and only one, value for each declared attribute. If the attributes color, shape, and weight are defined with appropriate domains, the complex

$$[\text{color} = \text{red}][\text{shape} = \text{round}][\text{weight} = 25]$$

would be an event. Events are also frequently referred to as *examples* since they serve as input examples for inductive learning algorithms.

Star

A *star* of an event e against a set of events F, denoted G(e | F), is the set of all possible maximally general complexes which cover event e and don't cover any of the events in F. Descriptions which cover an event can be thought of as generalisations of the event. Since in practice we are only interested in generalisations of an event which satisfy certain criteria, such as covering a large number of the other events in

the same class and brevity, a star is usually reduced in size by eliminating uninteresting generalizations.

Cover

A *cover*, for the purposes of this thesis, is a disjunction of complexes. A cover may be interpreted as a description of a class of objects. Two types of descriptions are especially significant in machine learning: discriminant descriptions and characteristic descriptions.

Discriminant Description

A *discriminant descriptions* is a description which distinguishes a class of objects from a fixed number of other object classes. It is typically the most general possible description which covers all members of a given class and no members of any other class.

Characteristic Description

A *characteristic description* is a description which distinguishes a class of objects from an infinite number of other object classes. It is the most specific possible description which covers all members of the class.

Lexicographic Evaluation Function (LEF)

A *lexicographic evaluation function with tolerances* is a list of criterion–tolerance pairs. A criterion is usually given as the name of a primitive evaluation function, and a tolerance is given as a real number between 0 and 1. A LEF can be used to establish a partial ordering on a set of objects, but we are usually interested only in the "best" subset of the set of objects with respect to the LEF. To find this subset, the first primitive evaluation function is applied to all members of a set of objects, and a fraction of the "best" objects as determined by the tolerance are retained. If the tolerance is 0, only those objects which tie with the "best" are retained. The process is repeated with the next criterion–tolerance pair on the set of retained objects until either the set of objects is small enough or the list of criterion–tolerance pairs is exhausted.

APPENDIX B: A USER'S GUIDE TO EXCEL

B.1. Introduction

This appendix is a reference manual for the ExceL program for learning VL_1 rules with exceptions. This program is useful for learning from noisy data, and for learning rules with legitimate exceptions, which may be summarised by an *unless* condition. The system is written in Franz Lisp [Foderaro, Sklower, and Layer, 1983] and runs under the Unix 4.2 operating system. The systems makes use of a package of macros and utilities designed to simplify the coding process [Becker, 1985a].

The system is designed for learning discriminant descriptions of classes of objects which are described in terms of attribute–value pairs. Some provisions have also been made for creating characteristic descriptions from discriminant descriptions. The decision classes may take several forms. The simplest form is a flat unordered set of decision classes, such as in a diagnosis problem where a single cause is assumed. The decision classes may also be hierarchically structured, such as in a taxonomy of species of animals. Simple linear decision classes are also supported, but this algorithm does not perform well in this case because it does not recognise patterns of concomitant variation.

The rules may be produced in either a single step or incrementally. The rules may be intersecting (ic), disjoint (dc) or sequentially ordered (vl). The rules may also be exact with respect to the set of provided training examples, or approximate. A conflict handling mechanism is also provided to assist the user in preparing consistent sets of training examples.

The system may be run either interactively or in batch mode. Interactive running is recommended only for small data sets (tens of events) since larger data sets may require considerable computation time for rule formation. Running the program consists of creating a data set, which contains not only the training events but also various control parameters, and then invoking the program. Results may be printed on the screen or directed to a file using the Unix system.

There are numerous parameters and options included in this system to allow for testing the effects of changes to the system. It was designed as a research tool without a great deal of concern for friendliness to naive users. However, defaults are provided for all parameters so that it is fairly easy to use the system

without a great deal of understanding of what the various parameters do.

B.2. Creating a Data Set

A data set is created using one of the standard editors available under the operating system (vi, emacs, etc.). List format is used for input to simplify parsing. Whenever a symbol or name is required, any legal LISP symbol may be used: a letter followed by any number of characters or numbers. A data set contains the following components:

```
(data_set_name
  parameters     (<parameters list>)
  types          (<attribute type declarations>)
  declarations   (<attribute declarations>)
  bgrules        (<background rules>)
  lisp           (<file names>)
  classes        <classification attribute>
  variables      (<event description attributes>)
  events         (<training events>)
  state          (<production system initial state>)
  statevars      (<variables used to describe state>))
```

The order of the components is *not* critical, however the spelling of the tag name for each component is. Briefly, parameters control various aspects of program execution, including preference criteria for candidate hypothesis selection. A type name may be declared to describe a type which is common to several attributes. Declarations declare attributes, their types, and optionally meta-information about the cost and relevance of an attribute. Background rules (bgrules) may be defined for creating attributes whose values are functionally dependent on the values of attributes used to describe the training events. The events will be partitioned into classes according to the domain of the attribute which is given as the classification attribute. A list of the attribute names used in the training events tells the system the correspondence between the list of values given for each training event and the appropriate attribute. The training events are used by the system to guide formulation of class descriptions. A sample data set which makes use of most of the available features is shown below.

A Sample Data Set

```
(animals

parameters
 ((maxstar 4)
  (trim nil)
  (utility 0.2)
  (confidence 0.8 )
  (lef ((max-newpromise 0.3)(min-selectors 0.2)(max-newposcovered 0.0))))

types  ((boole nom (yes no)))

declarations
 ((class             struc  ((birds <- water-birds land-birds)  mammals reptiles))
  (temp(blood)       nom    (warm cold) cost 1.3)
  (covering(body)    nom    (fur feathers scales skin))
  (habitat           nom    (air water land))
  (weight            lin    (1 .. 300 by 1 units lbs))
  (size              lin    (small medium large) cost 1.1 weight 0.7)
  (milk              boole  cost 2.0 weight 1.5)
  (eggs              boole  cost 1.5 weight 1.4))

bgrules
 (([size := large]      <-  [weight = 200 .. 300])
  ([size := medium]     <-  [weight = 100 .. 199])
  ([size := small]      <-  [weight = 1 .. 99]))

classes class

variables (NAME FREQ CONF class temp(blood) covering(body) eggs habitat milk weight)

events
 ((robin     10 0.9 land-birds  warm  feathers yes land  no   1)
  (emu        5  1.0 land-birds  warm  feathers yes land  no 100)
  (duck       8  0.9 water-birds warm  feathers yes water no  10)
  (penguin    7  0.8 water-birds warm  feathers yes water no  20)

  (human     20 1.0 mammals warm fur  no  land  yes 175)
  (dog        5  0.8 mammals warm fur  no  land  yes  50)
  (cat        5  0.9 mammals warm fur  no  land  yes  10)
  (platypus   1  0.8 mammals warm fur  yes water yes  20)
  (dolphin    2  0.9 mammals warm skin no  water no   300)

  (snake      4  1.0 reptiles cold scales yes land  no  3)
  (komodo     1  1.0 reptiles cold scales yes land  no 20)
  (alligator  3  0.9 reptiles cold scales yes water no 75)
  (frog       2  0.9 reptiles cold scales yes water no  1)))
```

The output generated by the ExceL program for this data set is shown below. The first set of timings is for initializing the data set. The second set of timings is for inducing rules from the initialized data. The table of parameter settings gives the values of all parameters for this particular run of the induction program. Each parameter is described in detail in the next section. For each class, a rule and some statistics are given. The format for a rule is:

$$\text{decision} \ <= \text{condition}_1 \vee \text{condition}_2 \vee \ldots \vee \text{condition}_n.$$

The decision is a single selector where the value specifies the decision class. Each condition is a conjunction or conjunction with an unless condition of the form:

$$S_1 \ S_2 \ \ldots \ S_n \ : \ (<\text{parameters}>)$$

or

$$S_1 \ S_2 \ \ldots \ S_n \ |\_ \ S_1 \ S_2 \ \ldots \ S_m \ : \ (<\text{parameters}>)$$

where each S_i is a selector.

The decision part of the rule occurs on a line by itself followed by a "<=" symbol which should be read as "if." Each condition occurs on a line by itself followed certain parameters. The values specified are, in order, the confidence assigned to the decision if the condition is satisfied, the number of newly covered positive training examples, the total number of positive training examples, and the number of covered negative training examples. Following each rule is another set of statistics about the rule as a whole. The complexity of rule is defined as the number of selectors plus the number of complexes plus the number of different variables in the rule. The rule complexity, number of complexes and number of selectors in the rule are listed, followed by the number of training examples for the class and the number of exceptional examples. Examples are counted by taking the sum of the confidence times the frequency associated with each example.

Sample Output

2.800 Sec. CPU
0.000 Sec. GC
16.000 Sec. REAL

Parameter Settings

mode	= ic	utility	= 0.2	stepsize	= all
maxstar	= 4	confidence	= 0.8	redo	= t
solutions	= 5	conflicts	= nil	trace	= 0
probe	= 1	gamma	= 2.0	unless	= t
trim	= nil	runconf	= 1.0		

print = (exceptions results stats summary params inittime runtime conflicts)

lef = ((max-newpromise 0.3) (min-selectors 0.2) (max-newposcovered 0.0))

2.433 Sec. CPU
0.000 Sec. GC
13.000 Sec. REAL

[class = birds] <=
    [covering(body) = feathers] : (0.89, 26.8, 26.8, 0.0)

Rule Complexity: 3
Rule Complexes: 1
Rule Selectors: 1
Class Examples: 26.80
Pos. Exceptions: 0.00
Neg. Exceptions: 0.00

[class = water-birds] <=
    [habitat = water] : (0.85, 12.8, 12.8, 0.0)

Rule Complexity: 3
Rule Complexes: 1
Rule Selectors: 1
Class Examples: 12.80
Pos. Exceptions: 0.00
Neg. Exceptions: 0.00

[class = land-birds] <=
    [habitat = land] : (0.93, 14.0, 14.0, 0.0)

Rule Complexity: 3
Rule Complexes: 1
Rule Selectors: 1
Class Examples: 14.00
Pos. Exceptions: 0.00
Neg. Exceptions: 0.00

[class = mammals] <=
[covering(body) = fur v skin] : (0.94, 31.1, 31.1, 0.0)

Rule Complexity: 3
Rule Complexes: 1
Rule Selectors: 1
Class Examples: 31.10
Pos. Exceptions: 0.00
Neg. Exceptions: 0.00

[class = reptiles] <=
[temp(blood) = cold] : (0.95, 9.5, 9.5, 0.0)

Rule Complexity: 3
Rule Complexes: 1
Rule Selectors: 1
Class Examples: 9.50
Pos. Exceptions: 0.00
Neg. Exceptions: 0.00

Each component of the data set will now be described in detail. Each section starts with a statement of the form or syntax for entering the component in the data set, followed by a description of the purpose of the component and the options available.

B.2.1. Parameters

Form: parameters ((param_name$_1$ <value>)(param_name$_2$ <value>) ...)

where each *param_name* is one of the parameter names listed below and *<value>* is a legal value for the associated parameter. The *parameters* list is used to select non-default values for program parameters. Most parameters take a single value. Certain parameters take a list of values in a particular format. Each parameter, its purpose, its allowed values, and its default value is described below.

mode

Specifies the mode to be used in cover generation. May be intersecting covers (ic), disjoint covers (dc), or sequential covers (vl).

Allowed Values:	ic, dc, vl
Default Value:	ic

maxstar

Specifies the maximum number of alternative descriptions retained during alternative description generation. A larger value causes better descriptions to be selected because the size of the search space is increased, but increases the amount of computation time required. A value which is roughly the number of attributes used to describe training events seems to work well.

Allowed Values:	Integer 1 .. $+\infty$
Default Value:	5

confidence

The minimum confidence that a description must have to be accepted as a solution. Low confidence solutions cover many negative events, high confidence solutions cover very few. In general, more computation is required to find high confidence solutions.

Allowed Values:	Float 0.0 .. 1.0
Default Value:	1.0

utility

Utility specifies the minimum utility that a complex may have and still be retained in the cover of a class. Requiring high utility forces positive exceptions to be generated. Changing utility causes no change in computational requirements.

Allowed Values:	Float 0.0 .. 1.0
Default Value:	0.1

solutions

Specifies the maximum number of solutions (descriptions which satisfy the confidence threshold) which will be searched for. The ADG algorithm terminates when either a certain number of solutions have been found or some solution has been found a certain number of probe steps have been tried.

Allowed Values:	Integer 1 .. $+\infty$
Default Value:	2

probe

Specifies the number of description generation cycles which will be performed after a solution is found in order to find another solution. If a larger value is used, more work will be done to find better solutions. A small value, 0 or 1, is usually adequate.

Allowed Values:	Integer 0 .. $+\infty$
Default Value:	1

trim

Specifies the method to be used for trimming complexes in a class cover. Trimming using the "drop" option reduces the number of values in the reference of selectors in a cover so that only those needed to cover the positive events actually covered by the complex are retained. The "unique" option reduces the number of values in the reference of selectors in a cover so that only those needed to cover the newly (uniquely) covered positive events are retained. The "extend" option extends the cover (making it more specialised) by multiplying in the refunion of covered events. This may be thought of as a characteristic description. If "nil" is used, the covers are not modified.

Allowed Values: nil, drop, unique, extend
Default Value: nil

conflicts

Specifies the method of conflict handling to be used. The "nil" option indicates that no action is to be taken. The "keep" option indicates that conflicting events are to be retained. The "drop" option indicates that conflicting events are to be dropped. The "max" option indicates that conflicting events are to be kept in the class where they occur the most frequently. The "ask" option indicates that the user wants to be asked to decide about each conflict individually.

Allowed Values: nil, keep, drop, max, ask
Default Value: nil

gamma

Specifies the minimum ratio of frequency of occurrence between the most frequent and next most frequent class when handling conflicts using "max". If the most frequent class is not gamma times bigger than the next biggest, then the conflicting event will be dropped from all classes.

Allowed Values: Float 0.0 .. $+\infty$
Default Value: 2.0

unless

Specifies that negative exceptions should be summarised in an *unless* condition.

Allowed Values: t, nil
Default Value: t

runconf

Specifies the confidence level required for a rule to fire when running the rule interpreter.

Allowed Values: Float 0.0 .. 1.0
Default Value: 1.0

stepsize

Specifies the stepsize (number of new training events considered per step) to be used during incremental learning.

Allowed Values: all, Integer 1 .. $+\infty$
Default Value: all

remember

Specifies how to remember training events during incremental learning. If "all" is used, all events will be remembered. If "wrong" is used, events will be remembered only if the are misclassified by the current hypothesis.

Allowed Values: all, wrong
Default Value: all

trace

Specifies what level of tracing information to print. At level 0, nothing is printed. At level 3, all tracing printouts are invoked.

Allowed Values: Integer 0 .. 3
Default Value: 0

print

The value of this parameter is a list of symbols indicating what to print during and after a run of the program.

Symbol	Effect
conflicts	*announce conflicting events*
exceptions	*print positive exceptions and unless conditions*
results	*print class covers*
stats	*print statistics with each complex of a cover*
summary	*print summary statistics for each cover*
params	*echo input parameters with the output (batch operation)*
verbose	*print periods to show the program is still running*
inittime	*print the CPU time for initialization*
runtime	*print the CPU time for cover generation*
printtime	*print the CPU time for displaying results*

Allowed Values: any subset of –
(conflicts exceptions results stats summary params verbose inittime runtime printtime)
Default Value: (exceptions results stats summary params inittime runtime conflicts)

lef

Specifies the evaluation functions which will be applied when selecting descriptions from among a list of alternatives. A LEF is a list of criterion–tolerance pairs. A criterion function may be selected from the list below. A tolerance must have a Float value between 0.0 and 1.0. A tolerance of 0.0 specifies that only those descriptions which are as good as the best description will be retained. A tolerance of 1.0 specifies that *all* descriptions will be retained.

Criterion	Effect
max–newpromise	*Maximize promise based on newly covered positive events.*
min–newpromise	*Minimize promise based on newly covered positive events.*
max–selectors	*Maximize the number of selectors in a complex.*
min–selectors	*Minimize the number of selectors in a complex.*
max–cost	*Maximize the cost of a complex.*
min–cost	*Minimize the cost of a complex.*
max–weight	*Maximize the weight of a complex.*
min–weight	*Minimize the weight of a complex.*
max–promise	*Maximize promise based on all covered positive events.*
min–promise	*Minimize promise based on all covered positive events.*
max–confidence	*Maximize estimated confidence.*
min–confidence	*Minimize estimated confidence.*
max–newposcovered	*Maximize number of newly covered positive events.*
min–newposcovered	*Minimize number of newly covered positive events.*
max–poscovered	*Maximize number of covered positive events.*
min–poscovered	*Minimize number of covered positive events.*
max–negcovered	*Maximize number of covered negative events.*
min–negcovered	*Minimize number of covered negative events.*

Allowed Values: A list of criterion–tolerance pairs.
Default Value: ((max–newpromise 0.0) (max–newposcovered 0.0) (min–cost 0.0)))))

repeat

Specifies a new parameter list for subsequent runs of the program when running in batch mode. Only parameter values which need to *changed* from the values used in the previous run need be specified. This allow one to run the program several times with different parameter settings in one batch run.

Allowed Values: A list of parameter lists.
Default Value: nil

The allowed and default values for all parameters are summarized in the table below.

Parameter	Allowed Values	Default Value
maxstar	Integer 1 .. +∞	5
confidence	Float 0.0 .. 1.0	1.0
utility	Float 0.0 .. 1.0	0.1
solutions	Integer 1 .. +∞	2
probe	Integer 0 .. +∞	1
trim	nil, drop, extend	nil
conflicts	nil, keep, drop, max	nil
gamma	Float 0.0 .. +∞	2.0
construct	t, nil	nil
runconf	Float 0.0 .. 1.0	1.0
mode	ic, dc, vl	ic
stepsize	all, Integer 1 .. +∞	all
remember	all, wrong	all
trace	Integer 0 .. 3	0
print	any subset of: (conflicts exceptions results stats summary params verbose inittime runtime printtime)	(exceptions results stats summary params inittime runtime conflicts)
lef	A list of criterion-tolerance pairs.	((max-newpromise 0.0) (min-selectors 0.0) (max-newposcovered 0.0)))))
repeat	A list of parameter lists.	nil

B.2.2. Declarations

Form: types (type_decl$_1$ type_decl$_2$ ...)

declarations (attr_decl$_1$ attr_decl$_2$ ...)

where each *type_decl* is a type declaration and each *attr_decl* is an attribute declaration, described below.

Type and attribute declarations are used to associate domains and other information with attributes. Type declarations are to be used to associate a name with a domain type that may be common to several attributes. A type declaration has the form:

(type_name domain_type members)

An attribute declaration has the form:

(attribute_name domain_type members [cost n][weight n][abbrev s][ask a])

or

(attribute_name type_name [cost n][weight n][abbrev s][ask a])

Type_name and *attribute_name* are names created by the user. These may be single symbols, or compound names of the form "symbol(symbol)". It is a good idea to pick meaningful names.

Domain_type must be one of *nom, lin, cyc,* or *struc*. The form of *members* depends on the domain type.

Domain Type	Abbreviation
nominal	nom
linear	lin
cyclic	cyc
structured	struc

In a nominal members list one simply gives a list of symbols representing the elements of the domain. A linear domain may consist of an ordered list of symbols or a numerical range with an optional stepsize and optional units. The default epsilon value (stepsize) is 1. All rules which apply to linear domains apply to cyclic domains. In a structured members list, each sublist represents a parent node to the left of the arrow and its children on the right of the arrow. A child may be a substructure as well as a symbol. In addition, a symbol may occur at the top level of a structure specification.

Additional information may associated with an attribute name:

cost
A value between 0.0 and $+\infty$, which indicates the cost associated with obtaining a value for an attribute in the real world. The default cost is 1.0.

weight
A value between 0.0 and $+\infty$, which indicates the relevance of this attribute with respect to the classification being made. The default weight is 1.0.

abbrev
An abbreviated name which may be used instead of the principle name in the variables list.

ask
A string describing the meaning of the attribute. This is printed to the user when asking for a value for the attribute when it occurs in a rule in an expert system (see section on rule running).

B.2.3. Background Rules

Form: bgrules (bg_rule_1 bg_rule_2 ...)

where each *bg_rule* is a background rule, described below. Background rules are used for generating selectors with values that are functionally dependent on the values of known attributes. This is useful when converting relational information to attribute form, and as a labor saving device. For example, if an object has attributes *shape, height,* and *width*, a rule may defined which generates a value for the attribute *area*. A background rule may either elaborate the description of the event by adding the new selector to the event, or replace one or more selectors with the new selector. The rules are distinguished by the type of arrow used:

Elaboration:	(l.h.s.	<<-	r.h.s.)
Replacement:	(l.h.s.	<-	r.h.s.)

where *l.h.s.* and *r.h.s.* are lists of selectors. When an event is processed using the background rules, each rule is applied to the event once in the sequence that the rules are given, thus a rule can use values which are generated by preceding rules. All variables used in background rules, whether initially in the event descriptions or generated via a background rule, must be declared.

Three types of selectors are used in background rules:

1. [Var = k]
2. [predicate]
3. [Var := f(x)]

All three types of selectors may occur on the l.h.s. of background rules. Only types 1 & 2 may occur on the r.h.s. of rules. The 3rd type is used to assign a value to an attribute and thus may only occur on the l.h.s. of a rule. If a predicate occurs on the l.h.s. of a rule, a boolean selector with a compound name is created.

Arbitrarily complex functions may be used for deriving the new values. Within the system, operators are defined for deriving values based on logical or arithmetic functions. Rules which use logical functions are called *L-rules*, and rules which use arithmetic functions are called *A-rules*. In addition, any LISP

function may be used. Provisions have been made for loading files with LISP function definitions when a data set is loaded (see the next section). The following operators have been defined for use in background rules:

Arithmetic Functions

Symbol	Description	Syntax	Precedence
-	unary minus	prefix	900
^	exponentiation	infix	870
!	factorial	postfix	850
*	multiplication	infix	800
/	division	infix	800
+	addition	infix	700
-	subtraction	infix	700
mod	modulo	infix	600

Arithmetic Predicates

Symbol	Description	Syntax	Precedence
>	greater than	infix	550
>=	greater or equal	infix	550
<	less than	infix	550
<=	less or equal	infix	550
=	equal	infix	500
<>	not equal	infix	500

Logical Operators

Symbol	Description	Syntax	Precedence
not	logical not	prefix	900
and	logical and	infix	850
or	logical or	infix	800
=>	implication	infix	750
<=>	equivalence	infix	750

Internal Disjunction

Symbol	Description	Syntax	Precedence
..	range operator	infix	950
v	disjunction	infix	900

Symbols with a higher precedence bind more tightly than those with a lower precedence. In general, symbols with equal precedence bind more tightly on the left. These operators are defined for use with a data

driven parser [Becker, 1985c] and it is relatively easy to define new operators for the system. When using LISP functions, the syntax is prefix form:

<function> (<argument_list>)

where <function> is the name of a defined LISP function, and <argument_list> is a list of constants and/or declared attribute names. The following examples show a number of the different forms which may be used for background rules. Note that all operators must be surrounded by blanks.

([size := small] <<- [weight = 0 .. 25])

This is a simple elaboration rule. The attribute *size* is assigned the value *small* when the value of the attribute *weight* is between 0 and 25 inclusive.

([area := length * height] <<-)

This is an elaboration rule which is always triggered since the r.h.s. is empty. The attribute *area* is assigned the value which is the product of the values of attributes *length* and *height*.

([color := purple] <- [color = red v blue])

This is a replacement rule which replaces the value of *color*.

([color := ask(color)] <- [height <> length][color = orange v purple])

This is a another replacement rule. The left hand side contains a selector which will assign a new value to the *color* attribute based on the result of evaluating the *ask* function, which is a user-defined LISP function. The r.h.s. contains a predicate [height <> length], and a set selector. Only the set selectors – those with "=" as the relation and a disjunction or range of values – are replaced. A predicate is used to determine when the replacement should occur.

B.2.4. Lisp Code Files

Form: lisp ($pathname_1$ $pathname_2$ ...)

where each *pathname* is the file system pathname for a file to be loaded. When user defined LISP functions are used in background rules, there must be a way to load the function definitions into the system. Each file listed in the *lisp* component will be loaded when the data set is initialized.

B.2.5. Classes

Form: classes attribute

The data set is partitioned into classes according to the values of the domain of an attribute singled out as the classification attribute. For nominal domains, each value in the domain characterises a class. For structured domains, each node characterises a class and a structured decision rule is generated. Linear decision classes are supported only for small integer and symbolic linear domains. For set valued domains, the values are the names of attributes used to define decision classes between which no discrimination is to be made. In general, declarations of decision classes may be nested in this way as long as unique names are used.

B.2.6. Training Examples

Form: variables (var_1 var_2 ...)

events ($event_1$ $event_2$ ...)

where each *var* is an attribute name, and each *event* is a training event. Each training event is a list of values for each of the variables in the *variables* component of the data set. Each variable must occur only once, and each event must have a value for each variable, in the correct order. The values must be legal values from the domain of the associated variable.

There are three special variables which may be used to associate meta-information with each event. These are NAME, CONF, and FREQ.

NAME

NAME is used to give a *unique* name to each event. This name will be used whenever the event is printed instead of the conjunction of selectors. This is of special interest when positive exceptions are generated. Any LISP symbol may be used as a name. There is no default name.

CONF

CONF is used to assign a confidence to each event which indicates how sure the expert who classified the event is that the event belongs to the associated decision class. Allowed values are 0.0 .. 1.0. The default confidence for an event is 1.

FREQ

FREQ is used to indicate the frequency of occurrence of an event. This eliminates the need to enter duplicate copies of an event if it occurs many times. Allowed values are integers greater than 1. The default frequency is 1.

B.2.7. State

Form: state ([var1 = val1][var2 = val2] ... [varn = valn]).

The state entry is used to specify the initial internal for the production system interpreter. It consists of a single VL_1 complex which has a single value for each variable. The initial state need not be specified, and may be omitted.

B.2.8. State Variables

Form: statevars (var1 var2 ... varn).

This entry is a list of the variables in the state complex. It should be a superset of the variables in the state complex, if one is given. This is needed since the initial values of some variables may be unknown.

B.3. Interactive Running

System invocation is done via an alias which may be defined in the users ".login" file. The alias definition is:

alias cover "lisp -f /mntb/2/michalski/jbecker/aq/cover.o \!*"

Alternatively, one may simply type the long form of the command:

lisp -f /mntb/2/michalski/jbecker/aq/cover.o

When the program is invoked with no command line arguments, it enters the interactive mode. Otherwise, it enters the batch mode described below. The system is largely menu driven with prompting for other input when needed. The top level menu looks like this:

ExceL Menu

E: Edit Data	Edit a data set file.
L: List Data	View a data set file.
G: Get Data	Loads a data set from a file.
B: Backup	Backup a data set to a file.
P: Parameters	Modify parameter settings.
C: Cover	Generate class covers.
S: Show Covers	Display the covers of a data set.
T: Test	Test covers against examples.
R: Run	Run a set of rules.
Q: Quit	Leave this menu.

Type the corresponding letter to invoke the function:

After each command completes the system pauses for a key press before returning to the top level menu (except for the *Quit* command). Also, on a LISP error, typing <control>-d one or several times will return the user to the top level menu. Each of the top level menu commands will now be described in detail.

E: Edit Data

This command allows the user to edit a data set without stopping the program run. When this command is selected, the system prompts for a file name. After the user enters a file pathname, the system executes the Unix "vi" command on the file. The user may then edit the file. When editing is completed, the file will be reloaded and the data set initialised.

L: List Data

This command allows the user to view the source file for a data set (or any other file). When this command is selected, the system prompts for a file name. After the user enters a file pathname, the system executes the Unix "more" command on the file.

G: Get Data

This command allows the user to load a data set into the lisp environment. Several data sets may be in the lisp environment at once, and the user may switch between them. When this command is selected, the system prompts for a file name. After the user enters a file pathname, the system loads

the file. The first atom in the list describing the data set is taken to be the data set name, and the data set is stored under this name. The data set is then initialized. This involves parsing various expressions, setting various parameters, and encoding various structures in a special internal form used by the program.

B: Backup

This command allows the user to save the state of data set on a file. This is most useful when doing incremental learning over a long period of time. When this command is selected, the system prompts for a data set name and a file name. After these are entered by the user, the system writes out information describing the data set to the named file. The file may be reload using the "Get Data" command.

P: Parameters

This command allows the user to modify parameter settings without editing the data set file. This is handy when experimenting with the effects of different parameter settings. The system will prompt for a data set name. After the user enters a name, the system will print the current settings for all system parameters, then prompt for a parameter name. After a parameter name is entered the system will prompt for a value. Once a value is entered, the system prints the new settings and the process repeats. When finished, type "ok" instead of a parameter name. Below is an example of what the screen looks like during the parameter setting process, user input is shown bold face:

Parameter Settings

mode	= ic	utility	= 0.0	stepsize	= all
maxstar	= 5	confidence	= 1.0	redo	= t
solutions	= 1	conflicts	= nil	trace	= 0
probe	= 0	gamma	= 2.0	unless	= t
trim	= nil	runconf	= 1.0		

print = (verbose exceptions results stats summary params inittime runtime conflicts)
lef = ((max-newpromise 0.3) (min-selectors 0.2) (max-newposcovered 0.0))

Parameter: mode
Value: dc

C: Cover

This command allows the user to invoke the cover generation process. When this command is invoked, the system will prompt for a data set name. After a name is entered, if the data set is present the system will invoke the learning algorithm on the data set. If the data set is not present, the system will look for a file with the given name and attempt to load it as a dataset. When finished, the results will be printed on the terminal screen.

S: Showcovers

This command allows the user to view the most recently generated set of covers for a data set. When this command is invoked, the system will prompt for a dataset name. After a name is entered, the system will display the set of covers for the data set according the print parameter.

T: Test

This command is used for generating a *confusion matrix* which shows how well a set of rules covers a set of examples (see example in section 5). When this command is invoked, the system will prompt for one data set containing rules (which must have previously been generated using the *Cover* command) and another data set containing training examples. The same data set may be used for both. The data sets must have identical variable declarations.

R: Run

This command allows the user to run a set of rules. The system prompts for a data set name, then invokes the rule runner on the named data set. Considerable user sophistication is needed to use this command because domain specific interface routines must be written for each application. The recommended method for developing a new application is to the distilling plant simulator as an example.

Q: Quit

This command causes the program to terminate and exit to the Unix operating system.

B.4. Batch Job Running

Running the program in batch mode is quite simple. The invocation command is identical to the one used for interactive running, except that additional command-line arguments specify the input and output files. The commmand syntax is:

cover <input_file> <output_file> &

The input file must be a properly structured data set. The output file is any file name the user chooses. The Unix "&" command may be used to put the job in background. The program will generate covers in the mode(s) specified in the parameters list and then exit.

APPENDIX C: EXPERIMENTAL DATA

C.1. Car Starting Example

The data set for the car starting example is given below, followed by program output for three different parameter settings. The first run is with parameters set to produce exact rules, the second run is with parameters set to produce approximate rules without unless conditions, and the third run is with parameters set to produce approximate rules with unless conditions.

Car Starting Data

```
(auto

    parameters
    ((mode dc)(utility 0.0)(confidence 1.0)(unless nil)
     (repeat
        (((mode dc)(utility 0.1)(confidence 0.9)(unless nil))
         ((mode dc)(utility 0.1)(confidence 0.9)(unless t)))))

    declarations
    ((engine      nom    (does_not_start starts))
     (action      nom    (none turn_key hot_wire))
     (gas_tank    lin    (empty filled))
     (battery     lin    (dead charged)))
    classes  engine
    variables
    (FREQ     engine          action      gas_tank    battery)

    events
    ((100     starts          turn_key    filled      charged)
     (1       starts          hot_wire    filled      charged)
     (1       does_not_start  turn_key    empty       charged)
     (1       does_not_start  turn_key    filled      dead)
     (100     does_not_start  none        filled      charged)
     (1       does_not_start  none        empty       charged)
     (1       does_not_start  none        filled      dead)
     (1       does_not_start  hot_wire    filled      dead)
     (1       does_not_start  hot_wire    empty       charged)))
```

Exact Rules

Parameter Settings

mode	= dc	utility	= 0.0	stepsize	= all
maxstar	= 5	confidence	= 1.0	redo	= t
solutions	= 5	conflicts	= nil	trace	= 0
probe	= 1	gamma	= 2.0	unless	= nil
trim	= nil	runconf	= 1.0		

print = (exceptions results stats summary params inittime runtime conflicts)
lef = ((max-newpromise 0.0) (max-newposcovered 0.0) (min-cost 0.0))

1.633 Sec. CPU
0.000 Sec. GC
6.000 Sec. REAL

[engine = does_not_start] <=
[action = none] : (1.00, 102, 102, 0) v
[gas_tank = empty] : (1.00, 2, 3, 0) v
[battery = dead] : (1.00, 2, 3, 0)

Rule Complexity: 9
Rule Complexes: 3
Rule Selectors: 3
Class Examples: 106
Pos. Exceptions: 0
Neg. Exceptions: 0

[engine = starts] <=
[action <> none][gas_tank = filled][battery = charged] : (1.00, 101, 101, 0)

Rule Complexity: 7
Rule Complexes: 1
Rule Selectors: 3
Class Examples: 101
Pos. Exceptions: 0
Neg. Exceptions: 0

parameters: (confidence, newly covered positive examples, total covered positive examples, covered negative examples)

Approximate Rules Without Unless Conditions

Parameter Settings

mode	= dc	utility	= 0.1	stepsize	= all
maxstar	= 5	confidence	= 0.9	redo	= t
solutions	= 5	conflicts	= nil	trace	= 0
probe	= 1	gamma	= 2.0	unless	= nil
trim	= nil	runconf	= 1.0		

print = (exceptions results stats summary params inittime runtime conflicts)
lef = ((max-newpromise 0.0) (max-newposcovered 0.0) (min-cost 0.0))

0.733 Sec. CPU
0.000 Sec. GC
1.000 Sec. REAL

[engine = does_not_start] <=
[action <> turn_key] : (0.99, 104, 104, 1)

Negative Exceptions:
[action = hot_wire][gas_tank = filled][battery = charged]

Positive Exceptions:
[action = turn_key][gas_tank = empty][battery = charged]
[action = turn_key][gas_tank = filled][battery = dead]

Rule Complexity: 3
Rule Complexes: 1
Rule Selectors: 1
Class Examples: 106
Pos. Exceptions: 2
Neg. Exceptions: 1

[engine = starts] <=
[action = turn_key] : (0.98, 100, 100, 2)

Negative Exceptions:
[action = turn_key][gas_tank = empty][battery = charged]
[action = turn_key][gas_tank = filled][battery = dead]

Positive Exceptions:
[action = hot_wire][gas_tank = filled][battery = charged]

Rule Complexity: 3
Rule Complexes: 1
Rule Selectors: 1
Class Examples: 101
Pos. Exceptions: 1
Neg. Exceptions: 2

parameters: (confidence, newly covered positive examples, total covered positive examples, covered negative examples)

Approximate Rules With Unless Conditions

Parameter Settings

mode	= dc	utility	= 0.1	stepsize	= all
maxstar	= 5	confidence	= 0.9	redo	= t
solutions	= 5	conflicts	= nil	trace	= 0
probe	= 1	gamma	= 2.0	unless	= t
trim	= nil	runconf	= 1.0		

print = (exceptions results stats summary params inittime runtime conflicts)
lef = ((max-newpromise 0.0) (max-newposcovered 0.0) (min-cost 0.0))

1.217 Sec. CPU
0.000 Sec. GC
4.000 Sec. REAL

[engine = does_not_start] <=
  [action <> turn_key] : (0.99, 104, 104, 1) |_
    [action = hot_wire][gas_tank = filled][battery = charged] : (1.00, 1, 1, 0)

Positive Exceptions:
  [action = turn_key][gas_tank = empty][battery = charged]
  [action = turn_key][gas_tank = filled][battery = dead]

Rule Complexity: 9
Rule Complexes: 2
Rule Selectors: 4
Class Examples: 106
Pos. Exceptions: 2
Neg. Exceptions: 1

[engine = starts] <=
  [action = turn_key] : (0.98, 100, 100, 2) |_
    [gas_tank = empty] : (1.00, 1, 1, 0) v
    [battery = dead] : (1.00, 1, 1, 0)

Positive Exceptions:
  [action = hot_wire][gas_tank = filled][battery = charged]

Rule Complexity: 9
Rule Complexes: 3
Rule Selectors: 3
Class Examples: 101
Pos. Exceptions: 1
Neg. Exceptions: 2

parameters: (confidence, newly covered positive examples, total covered positive examples, covered negative examples)

C.2. Data Sets

This section contains listings of the data sets described in Section 4.1.

C.2.1. Chemical Compound Domain

The first data set contains information about a sample of bimetalic coordination compounds. The goal is to find a set of rules which can be used to predict the metal to metal distance of a new compound, given a number of features of the compound. Although the system can produce correct rules for the given data, they are not satisfying to a human expert.

Chemical Compound Data

```
(chemistry

parameters ((confidence 0.9)(utility 0.1))

types
((ligand  struc ((X <- Br Cl I) (PR3 <- PEt3) (NR2 <- NMe2 NEt2) (R <- Me CH2SiR3) (OR  <- OCR3 OSiR3) CO NR3 none)))

declarations
 ((Metal            nom   (Co Cr Mn Mo Re Tc W))
  (Oxidation        lin   (-I O II III))
  (MMdistance       lin   (1.98 .. 3.123 by 0.001 units angstroms))
  (distance         nom   (very-near near far very-far))
  (Radius           lin   (1.25 .. 1.41  by 0.01  units angstroms))
  (Charge           lin   (-4 -2 0))
  (electrons/Metal  lin   (9 11 12 14 17))
  (Conformation     nom   (staggered eclipsed))
  (BondOrder        lin   (1 .. 4 by 1))
  (Ligand1 ligand)
  (Ligand2 ligand)
  (Ligand3 ligand)
  (Ligand4 ligand)
  (Ligand5 ligand))

bgrules
 (([distance := very-near] <- [MMdistance = 1.900  .. 2.200])
  ([distance := near] <- [MMdistance = 2.201  .. 2.500])
  ([distance := far] <- [MMdistance = 2.501  .. 2.800])
  ([distance := very-far] <- [MMdistance = 2.801  .. 3.200]))

classes distance

variables
 (NAME Metal Oxidation MMdistance Radius Charge electrons/Metal Conformation
   BondOrder Ligand1 Ligand2 Ligand3 Ligand4 Ligand5)

events (
("[Cr(CH3)8]4-"          Cr  II 1.980 1.29 -4 12 eclipsed  4 Me   Me   Me   Me   none)
("[Mo2Cl8]4-"            Mo  II 2.139 1.40 -4 12 eclipsed  4 Cl   Cl   Cl   Cl   none)
```

("|Mo(CH3)8|4-"	Mo	II	2.148	1.40	-4	12	eclipsed	4	Me	Me	Me	Me	none)
("Mo2Br4(p-MeC5H4N)4"	Mo	II	2.150	1.40	0	12	eclipsed	4	Br	NR3	Br	NR3	none)
("Mo2Cl4(p-MeC5H4N)4"	Mo	II	2.153	1.40	0	12	eclipsed	4	Cl	NR3	Cl	NR3	none)
("Mo2(CH2SiMe3)6"	Mo	III	2.167	1.40	0	9	staggered	3	CH2SiR3	CH2SiR3	CH2SiR3	none	none)
("Mo2(NMe2)4Cl2"	Mo	III	2.200	1.40	0	9	staggered	3	NMe2	NMe2	Cl	none	none)
("Mo2(NMe2)4Me2"	Mo	III	2.201	1.40	0	9	staggered	3	NMe2	NMe2	Me	none	none)
("Mo2(NMe2)6"	Mo	III	2.211	1.40	0	9	staggered	3	NMe2	NMe2	NMe2	none	none)
("Re2Cl6|PEt3|2"	Re	III	2.222	1.37	0	12	eclipsed	4	PEt3	Cl	Cl	Cl	none)
("Mo2(OCH2CMe3)6"	Mo	III	2.222	1.40	0	9	staggered	3	OCR3	OCR3	OCR3	none	none)
("Re2Cl4|P(Et)3|4"	Re	II	2.232	1.37	0	14	eclipsed	4	Cl	PEt3	Cl	PEt3	none)
("|Re2Cl8|2-"	Re	III	2.241	1.37	-2	12	eclipsed	4	Cl	Cl	Cl	Cl	none)
("Mo2(OSiMe3)6HNMe2"	Mo	III	2.242	1.40	0	11	staggered	3	OSiR3	OSiR3	OSiR3	NR3	none)
("W2|CH2Si(CH3)3|6"	W	III	2.255	1.41	0	9	staggered	3	CH2SiR3	CH2SiR3	CH2SiR3	none	none)
("|Re2Br8|2-"	Re	III	2.270	1.37	-2	12	eclipsed	4	Br	Br	Br	Br	none)
("W2(NMe2)4Cl2"	W	III	2.283	1.41	0	9	staggered	3	NMe2	NMe2	Cl	none	none)
("W2(NEt2)4Me2"	W	III	2.291	1.41	0	9	staggered	3	NEt2	NEt2	Me	none	none)
("W2(NMe2)6"	W	III	2.294	1.41	0	9	staggered	3	NMe2	NMe2	NMe2	none	none)
("W2(NEt2)4I2"	W	III	2.296	1.41	0	9	staggered	3	NEt2	NEt2	I	none	none)
("W2Cl2(NEt2)4"	W	III	2.301	1.41	0	9	staggered	3	Cl	NEt2	NEt2	none	none)
("W2(NEt2)4Br2"	W	III	2.303	1.41	0	9	staggered	3	NEt2	NEt2	Br	none	none)
("W2(OCHMe2)6(pyr)2"	W	III	2.332	1.41	0	11	staggered	3	OCR3	OCR3	OCR3	NR3	none)
("Co2(CO)6|P(n-Bu)3|2"	Co	O	2.665	1.25	0	17	staggered	1	CO	CO	CO	none	PEt3)
("Mn(CO)8|PEt3|2"	Mn	O	2.913	1.37	0	17	staggered	1	CO	CO	CO	CO	PEt3)
("Mn2(CO)10"	Mn	O	2.923	1.37	0	17	staggered	1	CO	CO	CO	CO	CO)
("Cr2(CO)10"	Cr	-I	2.970	1.29	-2	17	staggered	1	CO	CO	CO	CO	CO)
("Re2(CO)10"	Re	O	3.020	1.37	0	17	staggered	1	CO	CO	CO	CO	CO)
("Tc2(CO)10"	Tc	O	3.036	1.35	0	17	staggered	1	CO	CO	CO	CO	CO)
("Mo2(CO)10"	Mo	-I	3.123	1.40	-2	17	staggered	1	CO	CO	CO	CO	CO)))

C.2.2. Stenonema Mayfly Domain

This data set describes seven species of Stenonema mayfly nymphs in terms of salient physical characteristics. The goal is to find a set of rules which can be used to distinguish members of the different species. The system is able to find simple rules for performing this task.

Stenonema Mayfly Data

```
; STENONEMA MAYFLY NYMPH EVENTS
; see "Taxonomy and Ecology of Stenonema Mayflies"
;     Philip A. Lewis
;     National Environmental Research Center, USEPA
;     EPA-670/4-74-006 December 1974

(mayfly

parameters
 ((mode ic) (conflicts nil))

declarations
((class nom  (carolina candidum floridense gildersleevei interpunc minnentonka pallidum))
 (maxilla_crown_spines lin (0 .. 13 by 1))  ; Number of pectinate crowns on the maxilla (upper jaw)
 (maxilla_lateral_setae lin (0 .. 50 by 1))  ; Number of lateral setae (bristles) on the maxilla
 (inner_canine_teeth lin (0 .. 9 by 1))  ; Number of teeth on inner canine of left mandible
 (outer_canine_teeth lin (0 .. 9 by 1))  ; Number of teeth on inner edge of outer canine, left mandible
 (terga_mid_dorsal_pale_streaks nom (absent present))
 (terga_dark_posterior_margins nom (absent present))
 (dark_marks_sterna_9 nom (absent present)))

classes class

variables
 (class maxilla_crown_spines maxilla_lateral_setae inner_canine_teeth
  outer_canine_teeth terga_mid_dorsal_pale_streaks terga_dark_posterior_margins dark_marks_sterna_9)

events
  ((carolina     10   21   2   7   absent   absent   absent)
   (carolina     10   24   2   8   absent   absent   absent)
   (carolina     10   30   2   8   absent   absent   absent)
   (carolina     10   26   2   7   absent   absent   absent)
   (carolina     10   28   2   7   absent   absent   absent)
   (carolina     10   20   2   7   absent   absent   absent)
   (carolina     10   21   2   7   absent   absent   absent)
   (carolina     10   28   2   8   absent   absent   absent)
   (carolina     10   30   2   7   absent   absent   absent)
   (carolina     10   25   2   8   absent   absent   absent)
```

(candidum	7	15	0	8	present	absent	present)
(candidum	9	25	0	8	present	absent	present)
(candidum	9	17	0	8	present	absent	present)
(candidum	8	17	0	8	present	absent	present)
(candidum	9	19	0	8	present	absent	present)
(candidum	7	23	0	8	present	absent	present)
(candidum	8	20	0	8	present	absent	present)
(candidum	7	19	0	8	present	absent	present)
(candidum	7	18	0	8	present	absent	present)
(candidum	9	22	0	8	present	absent	present)
(floridense	8	20	4	7	present	absent	present)
(floridense	8	21	4	7	present	absent	present)
(floridense	8	22	4	7	present	absent	present)
(floridense	9	25	4	7	present	absent	present)
(floridense	9	24	4	7	present	absent	present)
(floridense	9	23	4	7	present	absent	present)
(floridense	8	23	4	7	present	absent	present)
(floridense	8	24	4	7	present	absent	present)
(floridense	9	20	4	7	present	absent	present)
(floridense	9	21	4	7	present	absent	present)
(floridense	8	24	4	7	present	absent	present)
(floridense	9	22	4	7	present	absent	present)
(floridense	8	25	4	7	present	absent	present)
(gildersleevei	11	31	3	7	present	absent	present)
(gildersleevei	12	45	4	9	present	absent	present)
(gildersleevei	13	35	5	9	present	absent	present)
(gildersleevei	12	39	7	9	present	absent	present)
(gildersleevei	12	42	5	8	present	absent	present)
(gildersleevei	12	36	5	9	present	absent	present)
(gildersleevei	13	30	3	9	present	absent	present)
(gildersleevei	11	31	6	9	present	absent	present)
(gildersleevei	11	10	7	7	present	absent	present)
(gildersleevei	13	36	6	8	present	absent	present)
(interpunc	8	24	2	6	present	present	absent)
(interpunc	8	26	5	5	present	present	absent)
(interpunc	10	30	4	6	present	present	absent)
(interpunc	9	22	4	5	present	present	absent)
(interpunc	9	29	2	7	present	present	absent)
(interpunc	10	20	4	7	present	present	absent)
(interpunc	9	27	5	7	present	present	absent)
(interpunc	10	28	5	7	present	present	absent)
(interpunc	10	30	5	6	present	present	absent)
(interpunc	8	23	2	5	present	present	absent)
(minnentonka	8	32	4	6	present	absent	present)
(minnentonka	10	36	4	6	present	absent	present)
(minnentonka	10	40	4	7	present	absent	present)
(minnentonka	9	30	4	6	present	absent	present)
(minnentonka	8	34	4	7	present	absent	present)

```
(minnentonka   8  37  4  7   present   absent   present)
(minnentonka   9  37  4  7   present   absent   present)
(minnentonka   9  30  4  6   present   absent   present)
(minnentonka   8  30  4  7   present   absent   present)
(minnentonka   9  37  4  6   present   absent   present)

 (pallidum    12  24  2  5   present   absent   present)
 (pallidum    11  24  2  8   present   absent   present)
 (pallidum    13  21  2  6   present   absent   present)
 (pallidum    11  25  2  5   present   absent   present)
 (pallidum    12  25  2  7   present   absent   present)
 (pallidum    11  23  2  8   present   absent   present)
 (pallidum    12  22  2  7   present   absent   present)
 (pallidum    11  20  2  5   present   absent   present)
 (pallidum    11  20  2  5   present   absent   present)
 (pallidum    13  21  2  6   present   absent   present)))
```

C.2.8. Freshwater Distilling Plant Domain

This data set contains training examples for operation of a simulation of the seawater system of a seawater to freshwater distilling plant. Each row in the "events" table is used as a training example five times - once for each of five valves in the system. The correct action for each valve is also given in each row. The set of 177 examples given here is sufficient to allow the system to learn adequate although not perfect rules for operating the distilling plant simulator.

Freshwater Distilling Plant Data

```
(evaps

parameters
 ((mode ic) (maxstar 20) (solutions 10)
  (lef ((max-newpromise 0.1)(min-cost 0.0))))

types
 ((level lin (0.0 .. 1000.0 by 0.1))
  (qlevel lin (low ok high))
  (change lin (-200.0 .. 200.0 by 0.1))
  (qchange lin (down steady up))
  (pump nom (on off))
  (adjust nom (open ok shut))
  (posn lin (0.0 .. 1.0 by 0.1)))

declarations (
 (adjust-v1 adjust) (adjust-v2 adjust) (adjust-v3 adjust) (adjust-v4 adjust) (adjust-v5 adjust)
 (action set (adjust-v1 adjust-v2 adjust-v3 adjust-v4 adjust-v5))
 (pump(p1) pump) (pump(p2) pump)
 (level(C1) level) (level(C2) level) (level(C3) level)
 (C1-level qlevel ) (C2-level qlevel ) (C3-level qlevel )
 (change(C1) change) (change(C2) change) (change(C3) change)
 (C1-change qchange ) (C2-change qchange ) (C3-change qchange )
 (position(v1) posn) (position(v2) posn) (position(v3) posn) (position(v4) posn) (position(v5) posn))

brules (
 ([C1-level := low] <- [level(C1) = 0.0 .. 450.0])
 ([C1-level := ok] <- [level(C1) = 450.1 .. 549.9])
 ([C1-level := high] <- [level(C1) = 550.0 .. 1000.0])
 ([C2-level := low] <- [level(C2) = 0.0 .. 450.0])
 ([C2-level := ok] <- [level(C2) = 450.1 .. 549.9])
 ([C2-level := high] <- [level(C2) = 550.0 .. 1000.0])
 ([C3-level := low] <- [level(C3) = 0.0 .. 450.0])
 ([C3-level := ok] <- [level(C3) = 450.1 .. 549.9])
```

```
(|C3-level := high| <- |level(C3) = 550.0 .. 1000.0|)

(|C1-change := down|    <- |change(C1) = -200.0 .. -4.9|)
(|C1-change := steady|  <- |change(C1) = -5.0 .. 5.0|)
(|C1-change := up|      <- |change(C1) = 5.1 .. 200.0|)

(|C2-change := down|    <- |change(C2) = -200.0 .. -4.9|)
(|C2-change := steady|  <- |change(C2) = -5.0 .. 5.0|)
(|C2-change := up|      <- |change(C2) = 5.1 .. 200.0|)

(|C3-change := down|    <- |change(C3) = -200.0 .. -4.9|)
(|C3-change := steady|  <- |change(C3) = -5.0 .. 5.0|)
(|C3-change := up|      <- |change(C3) = 5.1 .. 200.0|))

lisp ( /mntb/2/michalski/jbecker/simulate/start )

classes action

variables
 (adjust-v1 adjust-v2 adjust-v3 adjust-v4 adjust-v5 pump(p1) pump(p2)
  level(C1) level(C2) level(C3) change(C1) change(C2) change(C3)
  position(v1) position(v2) position(v3) position(v4) position(v5))

events (
(ok    ok    ok    ok    ok    on  on   368.6  921.8  266.3   34.9  -31.3  12.1  0.5  0.4  0.4  0.9  0.3)
(open  shut  open  shut  open  on  on   738.3   41.8  995.3   18.3   -8.2   4.3  0.2  0.7  0.1  0.8  0.1)
(shut  open  shut  open  shut  on  on   288.9  984.9  311.4   -4.3    4.9  -3.6  0.9  0.1  0.8  0.3  0.9)
(open  shut  open  open  shut  on  on   687.4   24.8  441.5    7.4  -10.2   1.5  0.3  0.7  0.1  0.2  0.6)
(open  open  open  ok    open  on  on   475.9  230.5  500.5   -8.4    0.0   3.3  0.1  0.4  0.7  0.7  0.6)
(shut  ok    shut  shut  ok    on  on   744.9  793.6  507.4  -16.6   -2.1   8.8  0.6  0.1  0.5  0.1  0.3)
(open  shut  open  ok    shut  on  on   696.6    9.6  448.7    9.3  -15.2   7.2  0.4  0.6  0.2  0.3  0.5)
(ok    shut  open  shut  shut  on  on   703.4    0.0  461.6    6.8   -9.6  12.9  0.5  0.5  0.3  0.3  0.4)
(ok    shut  open  shut  ok    on  on   703.4    0.0  473.4    0.0    0.0  11.8  0.5  0.4  0.4  0.2  0.3)
(ok    ok    open  ok    ok    on  on   692.1    0.0  476.0  -11.3    0.0   2.5  0.5  0.3  0.5  0.1  0.3)
(shut  open  shut  open  shut  on  on   282.5  993.5  281.5   -1.3    2.9 -16.0  0.7  0.3  0.6  0.5  0.7)
(shut  ok    shut  open  shut  on  on   291.0  989.7  266.4    8.5   -3.8 -15.0  0.6  0.4  0.5  0.6  0.6)
(ok    ok    ok    open  shut  on  on   309.8  975.3  256.0   18.8  -14.4 -10.5  0.5  0.4  0.4  0.7  0.5)
(ok    ok    ok    open  shut  on  on   333.7  953.1  254.2   24.0  -22.2  -1.8  0.5  0.4  0.4  0.8  0.4)
(shut  shut  ok    ok    ok    on  on   461.3  845.7  295.7   49.6  -40.2  15.6  0.6  0.4  0.4  0.9  0.3)
(open  ok    ok    ok    ok    on  on   411.7  886.0  280.1   43.1  -35.8  13.9  0.3  0.4  0.4  0.9  0.3)
(open  shut  ok    ok    ok    on  on   552.1  751.8  332.1   38.1  -49.2  19.1  0.4  0.2  0.4  0.9  0.3)
(ok    ok    ok    ok    ok    on  on   547.9  698.1  352.9   -4.2  -53.7  20.8  0.5  0.1  0.4  0.9  0.3)
(ok    open  ok    ok    ok    on  on   530.8  577.4  399.7  -10.3  -62.6  24.2  0.5  0.1  0.4  0.9  0.3)
(ok    shut  open  ok    ok    on  on   549.7  510.3  425.6   18.9  -67.1  26.0  0.5  0.2  0.4  0.9  0.3)
(ok    open  open  shut  ok    on  on   531.6  453.4  453.3  -18.1  -56.9  27.7  0.5  0.1  0.5  0.9  0.3)
(open  open  ok    ok    open  on  on   439.3  268.9  507.5   -4.1   19.6   3.5  0.3  0.6  0.8  0.7  0.1)
(open  open  open  ok    open  on  on   498.7  248.9  494.2  -24.4  -37.0   3.1  0.4  0.2  0.5  0.7  0.2)
(ok    shut  ok    ok    ok    on  on   467.0  289.3  511.2   27.7   20.4   3.7  0.5  0.7  0.8  0.7  0.2)
(ok    ok    ok    ok    ok    on  on   490.8  332.6  519.0    1.6   22.1   4.0  0.5  0.4  0.8  0.7  0.2)
(ok    shut  ok    ok    ok    on  on   499.0  355.5  523.2    8.2   22.9   4.1  0.5  0.4  0.8  0.7  0.2)
(ok    open  ok    ok    shut  on  on   480.8  403.7  531.9  -24.4   24.5   4.4  0.5  0.2  0.8  0.7  0.1)
(ok    shut  ok    ok    shut  on  on   491.7  429.0  536.5   10.9   25.3   4.6  0.5  0.3  0.8  0.7  0.1)
(ok    open  shut  ok    ok    on  on   463.4  455.1  541.2  -28.2   26.1   4.7  0.5  0.2  0.8  0.7  0.2)
```

(ok	shut	ok	ok	ok	on	on	498.7	455.1	546.1	35.2	0.0	4.6	0.5	0.3	0.7	0.7	0.3)
(ok	ok	ok	ok	ok	on	on	494.4	455.1	548.1	-4.3	0.0	4.8	0.5	0.2	0.7	0.7	0.3)
(ok	ok	shut	open	ok	on	on	492.7	485.8	525.6	-1.7	30.7	-25.5	0.5	0.2	0.7	0.5	0.3)
(ok	shut	open	shut	ok	on	on	529.7	454.2	530.9	37.1	-31.6	5.3	0.5	0.2	0.6	0.7	0.3)
(ok	ok	shut	open	ok	on	on	444.8	487.6	481.0	44.4	34.2	-28.5	0.5	0.3	0.7	0.5	0.3)
(ok	shut	open	shut	ok	on	on	513.9	452.5	486.9	69.1	-35.1	5.9	0.5	0.3	0.6	0.7	0.3)
(shut	open	open	open	ok	on	on	501.4	420.5	483.4	-142.7	3.1	-46.4	0.6	0.0	0.4	0.4	0.3)
(shut	open	open	open	ok	on	on	461.7	420.5	441.6	-10.7	0.0	-13.1	0.9	0.2	0.6	0.6	0.3)
(ok	open	open	shut	ok	on	on	434.6	420.5	463.7	-21.7	0.0	19.3	0.5	0.4	0.8	0.8	0.3)
(open	shut	open	shut	open	on	on	764.9	30.8	999.0	26.6	-11.0	3.7	0.3	0.6	0.2	0.7	0.2)
(open	shut	open	shut	ok	on	on	790.8	21.2	996.8	25.9	-9.6	-2.2	0.4	0.5	0.3	0.6	0.3)
(ok	shut	open	ok	ok	on	on	811.2	17.0	990.1	20.3	-4.2	-6.7	0.5	0.4	0.4	0.5	0.3)
(ok	shut	open	ok	ok	on	on	825.1	17.0	981.7	13.9	0.0	-8.4	0.5	0.3	0.5	0.5	0.3)
(ok	shut	ok	ok	ok	on	on	825.9	22.7	971.6	0.8	5.7	-10.1	0.5	0.2	0.6	0.5	0.3)
(ok	ok	ok	ok	ok	on	on	807.9	29.4	959.8	-17.9	6.7	-11.8	0.5	0.1	0.6	0.5	0.3)
(ok	ok	ok	ok	ok	on	on	502.7	500.0	500.1	2.7	0.0	0.1	0.5	0.3	0.7	0.7	0.3)
(ok	ok	ok	ok	ok	on	on	507.6	500.0	500.4	4.9	0.0	0.3	0.5	0.3	0.7	0.7	0.3)
(ok	shut	ok	ok	ok	on	on	514.9	500.0	500.9	7.3	0.0	0.4	0.5	0.3	0.7	0.7	0.3)
(ok	ok	ok	ok	ok	on	on	517.1	500.0	501.5	2.2	0.0	0.6	0.5	0.2	0.7	0.7	0.3)
(ok	ok	ok	ok	ok	on	on	519.7	500.0	502.2	2.6	0.0	0.7	0.5	0.2	0.7	0.7	0.3)
(ok	shut	open	shut	ok	on	on	990.4	320.0	750.1	0.4	0.0	0.1	0.5	0.2	0.7	0.7	0.3)
(ok	ok	open	shut	ok	on	on	982.5	323.4	748.7	-8.0	3.4	-1.5	0.5	0.1	0.8	0.6	0.3)
(ok	ok	ok	shut	ok	on	on	967.2	333.2	743.9	-15.3	9.8	-4.8	0.5	0.1	0.9	0.5	0.3)
(ok	open	shut	open	ok	on	on	290.4	920.0	150.1	0.4	0.0	0.1	0.5	0.2	0.7	0.7	0.3)
(ok	ok	shut	open	ok	on	on	296.9	916.6	152.1	6.5	-3.4	1.9	0.5	0.3	0.6	0.8	0.3)
(ok	ok	ok	ok	ok	on	on	309.7	906.4	157.3	12.8	-10.2	6.8	0.5	0.3	0.5	0.9	0.3)
(shut	ok	shut	shut	ok	on	on	493.7	774.5	800.1	-3.3	-3.5	0.1	0.8	0.0	0.2	0.7	0.3)
(shut	ok	ok	shut	ok	on	on	498.5	761.5	798.7	4.4	-13.0	-1.5	0.7	0.1	0.1	0.6	0.3)
(shut	shut	ok	shut	ok	on	on	507.4	744.9	793.6	8.8	-16.6	-2.1	0.6	0.1	0.1	0.5	0.3)
(open	open	shut	open	shut	on	on	441.5	687.4	24.8	1.5	7.4	-10.2	0.3	0.2	0.7	0.1	0.6)
(open	ok	shut	open	shut	on	on	448.7	696.6	9.6	7.2	9.3	-15.2	0.4	0.3	0.6	0.2	0.5)
(ok	shut	shut	open	shut	on	on	461.6	703.4	0.0	12.9	6.8	-9.6	0.5	0.3	0.5	0.3	0.4)
(ok	shut	shut	open	ok	on	on	473.4	703.4	0.0	11.8	0.0	0.0	0.5	0.2	0.4	0.4	0.3)
(ok	ok	ok	open	ok	on	on	476.0	692.1	0.0	2.5	-11.3	0.0	0.5	0.1	0.3	0.5	0.3)
(ok	ok	ok	open	ok	on	on	479.9	673.2	0.0	3.9	-18.9	0.0	0.5	0.1	0.3	0.6	0.3)
(ok	ok	ok	open	ok	on	on	483.5	645.7	4.7	3.6	-27.5	4.7	0.5	0.1	0.3	0.7	0.3)
(shut	open	open	shut	shut	on	on	311.4	288.9	984.9	-3.6	-4.3	4.9	0.9	0.3	0.1	0.8	0.9)
(shut	open	open	shut	shut	on	on	281.5	282.5	993.5	-16.0	-1.3	2.9	0.7	0.5	0.3	0.6	0.7)
(shut	open	ok	shut	shut	on	on	266.4	291.0	989.7	-15.0	8.5	-3.8	0.6	0.6	0.4	0.5	0.6)
(ok	open	ok	ok	shut	on	on	256.0	309.8	975.3	-10.5	18.8	-14.4	0.5	0.7	0.4	0.4	0.5)
(ok	open	ok	ok	shut	on	on	254.2	333.7	953.1	-1.8	24.0	-22.2	0.5	0.8	0.4	0.4	0.4)
(ok	ok	ok	ok	ok	on	on	266.3	368.6	921.8	12.1	34.9	-31.3	0.5	0.9	0.4	0.4	0.3)
(shut	ok	shut	ok	ok	on	on	295.7	461.3	845.7	15.6	49.6	-40.2	0.6	0.9	0.4	0.4	0.3)
(open	ok	ok	ok	ok	on	on	280.1	411.7	886.0	13.9	43.1	-35.8	0.3	0.9	0.4	0.4	0.3)
(open	ok	shut	ok	ok	on	on	332.1	552.1	751.8	19.1	38.1	-49.2	0.4	0.9	0.2	0.4	0.3)
(ok	ok	ok	ok	ok	on	on	352.9	547.9	698.1	20.8	-4.2	-53.7	0.5	0.9	0.1	0.4	0.3)
(ok	ok	open	ok	ok	on	on	399.7	530.8	577.4	24.2	-10.3	-62.6	0.5	0.9	0.1	0.4	0.3)
(ok	ok	shut	open	ok	on	on	425.6	549.7	510.3	26.0	18.9	-67.1	0.5	0.9	0.2	0.4	0.3)
(ok	shut	open	open	ok	on	on	453.3	531.6	453.4	27.7	-18.1	-56.9	0.5	0.9	0.1	0.5	0.3)
(open	ok	open	open	open	on	on	500.5	475.9	230.5	3.3	-8.4	0.0	0.1	0.7	0.4	0.7	0.0)
(open	ok	open	ok	open	on	on	507.5	439.3	268.9	3.5	-4.1	19.6	0.3	0.7	0.6	0.8	0.1)
(open	ok	open	open	open	on	on	494.2	498.7	248.9	3.1	-24.4	-37.0	0.4	0.7	0.2	0.5	0.2)
(ok	ok	shut	ok	ok	on	on	511.2	467.0	289.3	3.7	27.7	20.4	0.5	0.7	0.7	0.8	0.3)

(ok	ok	ok	ok	ok	on	on	519.0	490.8	332.6	4.0	1.6	22.1	0.5	0.7	0.4	0.8	0.3)
(ok	ok	shut	ok	ok	on	on	523.2	499.0	355.5	4.1	8.2	22.9	0.5	0.7	0.4	0.8	0.3)
(ok	ok	open	ok	shut	on	on	531.9	480.8	403.7	4.4	-24.4	24.5	0.5	0.7	0.2	0.8	0.3)
(ok	ok	shut	ok	shut	on	on	536.5	491.7	429.0	4.6	10.9	25.3	0.5	0.7	0.3	0.8	0.3)
(ok	ok	open	shut	ok	on	on	541.2	463.4	455.1	4.7	-28.2	26.1	0.5	0.7	0.2	0.8	0.3)
(ok	ok	shut	ok	ok	on	on	546.1	498.7	455.1	4.6	35.2	0.0	0.5	0.7	0.3	0.7	0.3)
(ok	ok	ok	ok	ok	on	on	548.1	494.4	455.1	4.8	-4.3	0.0	0.5	0.7	0.2	0.7	0.3)
(ok	open	ok	shut	ok	on	on	525.6	492.7	485.8	-25.5	-1.7	30.7	0.5	0.6	0.2	0.7	0.3)
(ok	shut	shut	open	ok	on	on	530.9	529.7	454.2	5.3	37.1	-31.6	0.5	0.7	0.2	0.6	0.3)
(ok	open	ok	shut	ok	on	on	481.0	444.8	487.6	-28.5	44.4	34.2	0.5	0.6	0.3	0.7	0.3)
(ok	shut	shut	open	ok	on	on	486.9	513.9	452.5	5.9	69.1	-35.1	0.5	0.7	0.3	0.6	0.3)
(shut	open	open	open	ok	on	on	483.4	501.4	420.5	-46.4	-142.7	3.1	0.6	0.4	0.0	0.4	0.3)
(shut	open	open	open	ok	on	on	441.6	461.7	420.5	-13.1	-10.7	0.0	0.9	0.6	0.2	0.6	0.3)
(ok	shut	open	open	ok	on	on	463.7	434.6	420.5	19.3	-21.7	0.0	0.5	0.8	0.4	0.8	0.3)
(open	shut	shut	open	open	on	on	995.3	738.3	41.8	4.3	18.3	-8.2	0.2	0.8	0.7	0.1	0.1)
(open	shut	shut	open	open	on	on	999.0	764.9	30.8	3.7	26.6	-11.0	0.3	0.7	0.6	0.2	0.2)
(open	shut	shut	open	ok	on	on	996.8	790.8	21.2	-2.2	25.9	-9.6	0.4	0.6	0.5	0.3	0.3)
(ok	ok	shut	open	ok	on	on	990.1	811.2	17.0	-6.7	20.3	-4.2	0.5	0.5	0.4	0.4	0.3)
(ok	ok	shut	open	ok	on	on	981.7	825.1	17.0	-8.4	13.9	0.0	0.5	0.5	0.3	0.5	0.3)
(ok	ok	shut	ok	ok	on	on	971.6	825.9	22.7	-10.1	0.8	5.7	0.5	0.5	0.2	0.6	0.3)
(ok	ok	ok	ok	ok	on	on	959.8	807.9	29.4	-11.8	-17.9	6.7	0.5	0.5	0.1	0.6	0.3)
(ok	ok	ok	ok	ok	on	on	500.1	502.7	500.0	0.1	2.7	0.0	0.5	0.7	0.3	0.7	0.3)
(ok	ok	ok	ok	ok	on	on	500.4	507.6	500.0	0.3	4.9	0.0	0.5	0.7	0.3	0.7	0.3)
(ok	ok	shut	ok	ok	on	on	500.9	514.9	500.0	0.4	7.3	0.0	0.5	0.7	0.3	0.7	0.3)
(ok	ok	ok	ok	ok	on	on	501.5	517.1	500.0	0.6	2.2	0.0	0.5	0.7	0.2	0.7	0.3)
(ok	ok	ok	ok	ok	on	on	502.2	519.7	500.0	0.7	2.6	0.0	0.5	0.7	0.2	0.7	0.3)
(ok	shut	shut	open	ok	on	on	750.1	990.4	320.0	0.1	0.4	0.0	0.5	0.7	0.2	0.7	0.3)
(ok	shut	ok	open	ok	on	on	748.7	982.5	323.4	-1.5	-8.0	3.4	0.5	0.6	0.1	0.8	0.3)
(ok	shut	ok	ok	ok	on	on	743.9	967.2	333.2	-4.8	-15.3	9.8	0.5	0.5	0.1	0.9	0.3)
(ok	open	open	shut	ok	on	on	150.1	290.4	920.0	0.1	0.4	0.0	0.5	0.7	0.2	0.7	0.3)
(ok	open	ok	shut	ok	on	on	152.1	296.9	916.6	1.9	6.5	-3.4	0.5	0.8	0.3	0.6	0.3)
(ok	ok	ok	ok	ok	on	on	157.3	309.7	906.4	6.8	12.8	-10.2	0.5	0.9	0.3	0.5	0.3)
(shut	shut	ok	shut	ok	on	on	800.1	493.7	774.5	0.1	-3.3	-3.5	0.8	0.7	0.0	0.2	0.3)
(shut	shut	ok	ok	ok	on	on	798.7	498.5	761.5	-1.5	4.4	-13.0	0.7	0.6	0.1	0.1	0.3)
(shut	shut	shut	ok	ok	on	on	793.6	507.4	744.9	-2.1	8.8	-16.6	0.6	0.5	0.1	0.1	0.3)
(open	open	open	shut	shut	on	on	24.8	441.5	687.4	-10.2	1.5	7.4	0.3	0.1	0.2	0.7	0.3)
(open	open	ok	shut	shut	on	on	9.6	448.7	696.6	-15.2	7.2	9.3	0.4	0.2	0.3	0.6	0.3)
(ok	open	shut	shut	shut	on	on	0.0	461.6	703.4	-9.6	12.9	6.8	0.5	0.3	0.3	0.5	0.4)
(ok	open	shut	shut	ok	on	on	0.0	473.4	703.4	0.0	11.8	0.0	0.5	0.4	0.2	0.4	0.3)
(ok	open	ok	ok	ok	on	on	0.0	476.0	692.1	0.0	2.5	-11.3	0.5	0.5	0.1	0.3	0.3)
(ok	open	ok	ok	ok	on	on	0.0	479.9	673.2	0.0	3.9	-18.9	0.5	0.6	0.1	0.3	0.3)
(ok	open	ok	ok	ok	on	on	4.7	483.5	645.7	4.7	3.6	-27.5	0.5	0.7	0.1	0.3	0.3)
(shut	shut	open	open	shut	on	on	984.9	311.4	288.9	4.9	-3.6	-4.3	0.9	0.8	0.3	0.1	0.3)
(shut	shut	open	open	shut	on	on	993.5	281.5	282.5	2.9	-16.0	-1.3	0.7	0.6	0.5	0.3	0.7)
(shut	shut	open	ok	shut	on	on	989.7	266.4	291.0	-3.8	-15.0	8.5	0.6	0.5	0.6	0.4	0.3)
(ok	ok	open	ok	shut	on	on	975.3	256.0	309.8	-14.4	-10.5	18.8	0.5	0.4	0.7	0.4	0.5)
(ok	ok	open	ok	shut	on	on	953.1	254.2	333.7	-22.2	-1.8	24.0	0.5	0.4	0.8	0.4	0.4)
(ok	ok	ok	ok	ok	on	on	921.8	266.3	368.6	-31.3	12.1	34.9	0.5	0.4	0.9	0.4	0.3)
(shut	ok	ok	shut	ok	on	on	845.7	295.7	461.3	-40.2	15.6	49.6	0.6	0.4	0.9	0.4	0.3)
(open	ok	ok	ok	ok	on	on	886.0	280.1	411.7	-35.8	13.9	43.1	0.3	0.4	0.9	0.4	0.3)
(open	ok	ok	shut	ok	on	on	751.8	332.1	552.1	-49.2	19.1	38.1	0.4	0.4	0.9	0.2	0.3)
(ok	ok	ok	ok	ok	on	on	698.1	352.9	547.9	-53.7	20.8	-4.2	0.5	0.4	0.9	0.1	0.3)
(ok	ok	ok	open	ok	on	on	577.4	399.7	530.8	-62.6	24.2	-10.3	0.5	0.4	0.9	0.1	0.3)

(ok	open	ok	shut	ok	on	on	510.3	425.6	549.7	-67.1	26.0	18.9	0.5	0.4	0.9	0.2	0.3)
(ok	open	shut	open	ok	on	on	453.4	453.3	531.6	-56.9	27.7	-18.1	0.5	0.5	0.9	0.1	0.3)
(open	open	ok	open	open	on	on	230.5	500.5	475.9	0.0	3.3	-8.4	0.1	0.7	0.7	0.4	0.0)
(open	ok	ok	open	open	on	on	268.9	507.5	439.3	19.6	3.5	-4.1	0.3	0.8	0.7	0.6	0.1)
(open	open	ok	open	open	on	on	248.9	494.2	498.7	-37.0	3.1	-24.4	0.4	0.5	0.7	0.2	0.2)
(ok	ok	ok	shut	ok	on	on	289.3	511.2	467.0	20.4	3.7	27.7	0.5	0.8	0.7	0.7	0.3)
(ok	ok	ok	ok	ok	on	on	332.6	519.0	490.8	22.1	4.0	1.6	0.5	0.8	0.7	0.4	0.3)
(ok	ok	ok	shut	ok	on	on	355.5	523.2	499.0	22.9	4.1	8.2	0.5	0.8	0.7	0.4	0.3)
(ok	ok	ok	open	shut	on	on	403.7	531.9	480.8	24.5	4.4	-24.4	0.5	0.8	0.7	0.2	0.5)
(ok	ok	ok	shut	shut	on	on	429.0	536.5	491.7	25.3	4.6	10.9	0.5	0.8	0.7	0.3	0.4)
(ok	shut	ok	open	ok	on	on	455.1	541.2	463.4	26.1	4.7	-28.2	0.5	0.8	0.7	0.2	0.3)
(ok	ok	ok	shut	ok	on	on	455.1	546.1	498.7	0.0	4.6	35.2	0.5	0.7	0.7	0.3	0.3)
(ok	ok	ok	ok	ok	on	on	455.1	548.1	494.4	0.0	4.8	-4.3	0.5	0.7	0.7	0.2	0.3)
(ok	shut	open	ok	ok	on	on	485.8	525.6	492.7	30.7	-25.5	-1.7	0.5	0.7	0.6	0.2	0.3)
(ok	open	shut	shut	ok	on	on	454.2	530.9	529.7	-31.6	5.3	37.1	0.5	0.6	0.7	0.2	0.3)
(ok	shut	open	ok	ok	on	on	487.6	481.0	444.8	34.2	-28.5	44.4	0.5	0.7	0.6	0.3	0.3)
(ok	open	shut	shut	ok	on	on	452.5	486.9	513.9	-35.1	5.9	69.1	0.5	0.6	0.7	0.3	0.3)
(shut	open	open	open	ok	on	on	420.5	483.4	501.4	3.1	-46.4	-142.7	0.6	0.4	0.4	0.0	0.3)
(shut	open	open	open	ok	on	on	420.5	441.6	461.7	0.0	-13.1	-10.7	0.9	0.6	0.6	0.2	0.3)
(ok	open	shut	open	ok	on	on	420.5	463.7	434.6	0.0	19.3	-21.7	0.5	0.8	0.8	0.4	0.3)
(open	open	shut	shut	open	on	on	41.8	995.3	738.3	-8.2	4.3	18.3	0.2	0.1	0.8	0.7	0.1)
(open	open	shut	shut	open	on	on	30.8	999.0	764.9	-11.0	3.7	26.6	0.3	0.2	0.7	0.6	0.2)
(open	open	shut	shut	ok	on	on	21.2	996.8	790.8	-9.6	-2.2	25.9	0.4	0.3	0.6	0.5	0.3)
(ok	open	ok	shut	ok	on	on	17.0	990.1	811.2	-4.2	-6.7	20.3	0.5	0.4	0.5	0.4	0.3)
(ok	open	ok	shut	ok	on	on	17.0	981.7	825.1	0.0	-8.4	13.9	0.5	0.5	0.5	0.3	0.3)
(ok	ok	ok	shut	ok	on	on	22.7	971.6	825.9	5.7	-10.1	0.8	0.5	0.6	0.5	0.2	0.3)
(ok	ok	ok	ok	ok	on	on	29.4	959.8	807.9	6.7	-11.8	-17.9	0.5	0.6	0.5	0.1	0.3)
(ok	ok	ok	ok	ok	on	on	500.0	500.1	502.7	0.0	0.1	2.7	0.5	0.7	0.7	0.3	0.3)
(ok	ok	ok	ok	ok	on	on	500.0	500.4	507.6	0.0	0.3	4.9	0.5	0.7	0.7	0.3	0.3)
(ok	ok	ok	shut	ok	on	on	500.0	500.9	514.9	0.0	0.4	7.3	0.5	0.7	0.7	0.3	0.3)
(ok	ok	ok	ok	ok	on	on	500.0	501.5	517.1	0.0	0.6	2.2	0.5	0.7	0.7	0.2	0.3)
(ok	ok	ok	ok	ok	on	on	500.0	502.2	519.7	0.0	0.7	2.6	0.5	0.7	0.7	0.2	0.3)
(ok	open	shut	shut	ok	on	on	320.0	750.1	990.4	0.0	0.1	0.4	0.5	0.7	0.7	0.2	0.3)
(ok	open	shut	ok	ok	on	on	323.4	748.7	982.5	3.4	-1.5	-8.0	0.5	0.8	0.6	0.1	0.3)
(ok	ok	shut	ok	ok	on	on	333.2	743.9	967.2	9.8	-4.8	-15.3	0.5	0.9	0.5	0.1	0.3)
(ok	shut	open	open	ok	on	on	920.0	150.1	290.4	0.0	0.1	0.4	0.5	0.7	0.7	0.2	0.3)
(ok	shut	open	ok	ok	on	on	916.6	152.1	296.9	-3.4	1.9	6.5	0.5	0.6	0.8	0.3	0.3)
(ok	ok	ok	ok	ok	on	on	906.4	157.3	309.7	-10.2	6.8	12.8	0.5	0.5	0.9	0.3	0.3)
(shut	shut	shut	ok	ok	on	on	774.5	800.1	493.7	-3.5	0.1	-3.3	0.8	0.2	0.7	0.0	0.3)
(shut	ok	shut	ok	ok	on	on	761.5	798.7	498.5	-13.0	-1.5	4.4	0.7	0.1	0.6	0.1	0.3)
(ok	ok	open	ok	ok	on	on	673.2	0.0	479.9	-18.9	0.0	3.9	0.5	0.3	0.6	0.1	0.3)
(ok	ok	open	ok	ok	on	on	645.7	4.7	483.5	-27.5	4.7	3.6	0.5	0.3	0.7	0.1	0.3)

))

C.2.4. Soybean Disease Domain

This data set contains examples of seventeen different soybean diseases described in terms of characteristics of individual soybean plants as well as macroscopic information such as cropping history for the affected field and the distribution of diseased plants in the field. The goal is to generate rules for determining what disease a new example of a diseased soybean plant has.

Soybean Disease Data

```
(plant

parameters ((mode ic) (conflicts nil))

declarations
((class nom
        (alternaria anthracnose bact_blight bact_pustule brown_spot brown_stem charcoal_rot cyst_nematode diap_pod
       diap_stem downy_mildew frog_eye phyllosticta phytophthora powdery_mildew purple_seed rhizoctonia))

; ********** General and Environmental descriptors **********

(time_of_occurrence lin (april may june july august september october) abbrev x1)

 ; This question concerns the month of the year that the observed problem occurred. If the problem exists
 ; at the present time, chose the current month. If you are unsure exactly what month the problem manifested
 ; itself, or if the date falls between two months, just select your best guess.

(precipitation lin (below_normal normal above_normal) abbrev x2)

 ; This question refers to the amount of precipitation observed through the growing season up to the point
 ; where the crop problem was observed. If there is any doubt as to the level of precipitation, select "normal."

(temperature lin (below_normal normal above_normal) abbrev x3)

 ; Temperature refers to the average ambient temperature. You can compute this from common weather report
 ; data (degree-days) if you really have no idea what it is. Divide the number of degree-days by the number
 ; of days since May 1. If the result is less than 20, select "below normal." If
 ; the result is greater than 30, select "above normal." Otherwise, select "normal."

(cropping_history lin (none one two three four_or_more) abbrev x4)

 ; Some soybean diseases are especially common in fields that have produced a soybean crop for more than one
 ; season. If this is the first season that soybeans have been planted in the field where the problem was
 ; observed, select "none." If this is the second year that you have planted soybeans in the field,
 ; select "one," and so on. If more than one field is concerned, select that answer which corresponds
 ; to the field that has had a soybean crop for the least number of years.
```

```
(damaged_area
 nom (groups_low_areas groups_upland_areas whole_fields scattered_plants gradient_from_field_edge) abbrev x5)

; This question concerns the location of the damaged plants in the fields, and how the damage is occurring
; among various plants.  If more than one answer applies (i.e. the problem is manifesting itself in different ways)
; then select "scattered plants."

(severity lin (minor potentially_severe severe) abbrev x6)

; Try to take into account both the number of plants affected and the condition of the diseased plants.
; A problem could be severe just because a large number of plants are affected.

(plant_height nom (less_than_normal normal) abbrev x7)

; Plant height should not be confused with plant stand.  Height refers to the physical length of the plant.
; Stand refers to the fact that a plant may be erect or leaning.

; ****************************** Leaf descriptors ******************************
(condition_of_leaves nom (abnormal normal) abbrev x8)

; Leaves are abnormal if they are discolored, if they have holes in them,
; if they are shredded, if they are twisted, or if they have spots on them.

(leaf_spots nom (does_not_apply absent present) abbrev x9)

; Leaf spots may be small dots are large blotches, often of an off color, that
; occur on the leaves of diseased plants.  If any type of spot is occuring, specify "present."

(leaf_spot_color nom
 (does_not_apply none grey tan light_brown tan_with_dark_brown_border brown
  reddish_brown dark_brown_or_black purple yellow silver) abbrev x10)

; This refers to the color of the interior of the leaf spot, not the margin.  If more than one type of
; leaf spot is present, select that color that occurs on the most predominant type of spot.  If the
; color of the spots does not match any of the colors given, select the closest color.

(color_of_spot_on_reverse_side nom
 (does_not_apply none grey tan light_brown light_brown_with_dark_brown_border
  brown reddish brown dark_brown_or_black purple yellow) abbrev x11)

; If the leaf spots are not visible from the underside of the leaf, select "none."
; If they are, select that color which is closest to the color of the most spots visible.

(yellow_leaf_spot_halos nom (does_not_apply absent present) abbrev x12)

; Do not confuse halos with water-soaked margins.  Halos are opaque,
; while margins are translucent.  Halos are almost always yellow.

(margin_of_leaf_spots nom (does_not_apply water_soaked not_water_soaked) abbrev x13)

; Water_soaked margins are translucent rings around leaf spots.  Do not confuse these with halos,
; which are opaque and usually yellow.  Margins are generally colorless.
```

```
(raised_leaf_spots nom (does_not_apply absent present) abbrev x14)

; This question refers to the underside of the leaf.  Often, a spot occurs only on the upper side
; of leaves.  If, however, you can feel a bump or see a slight peak on the underside of the leaf
; where there is a spot on the upper side, then the leaf spots are raised.

(leaf_spot_growth nom
 (does_not_apply from_edge_of_leaf_inward with_concentric_rings spread_out_and_plain
  brown_veinal_necrosis necrosis_across_veins vein_delimited_necrosis) abbrev x15)

; To answer this question, consider the continuity and location of the spots on the leaf.  If the spots are heaviest at the edge or
; the tip, then select "from edge of leaf inward."  If each spot seems to have swirls, then select "scattered with concentric rings."
; If the spots occur along the leaf veins, then select "veinal necrosis."  In any other case, select "scattered and plain."

(leaf_spot_size
 lin (does_not_apply thirty_secondth_to_sixteenth_inch sixteenth_to_eighth_inch greater_than_eighth_inch) abbrev x16)

; This seems pretty obvious; you must be the person who asked what hail is.  Leaf spots have a certain diameter:
;       the itty-bitty baby ones are less than 1/16"
;       the mommy ones are 1/8" - 1/16"
;       the great big daddy ones are greater than 1/8"
;       the really huge, monstrous ones covering the entire leaf are whole leaf

(shot_holing nom (does_not_apply absent present) abbrev x17)

; Shot holing is a process whereby a leaf spot eventually falls out and a hole remains in the leaf where the spot
; had been.  Do not confuse shot holes with insect damage.  Shot holes are generally perfectly symmetric circles.

(shredding nom (does_not_apply absent present) abbrev x18)

; Shredding is similar to shot holing.  It appears as lacerations in the affected leaves.

(leaf_malformation nom (does_not_apply absent present) abbrev x19)

; This question concerns the general SHAPE of the leaves.  If the leaves are twisted or contorted, then leaf malformation is present.

(premature_defoliation nom (does_not_apply absent present attached_but_dead_leaves) abbrev x20)

; Premature defoliation is present if the leaves are so devastated by disease that they are falling off or are completely dead.

(leaf_mildew_growth nom (does_not_apply absent on_upper_leaf_surface on_lower_leaf_surface) abbrev x21)

; Mildew is generally a white spot that may look like mold.  The spots may
; be very small, so both sides of the leaf should be checked carefully.

(leaf_discoloration nom
 (does_not_apply none general_yellowing greenish_yellowing mottling bronze
  purple nondescript yellowed_from_margins_or_interveinal) abbrev x22)

; This question refers to the color of the leaves apart from any leaf spots.  Mottling refers to light or dark green discoloration
; around the veins.  Discoloration around shreading or shot-holing should not be considered when answering this question.

(position_of_affected_leaves nom (does_not_apply near_lower_nodes near_upper_nodes scattered_on_plant) abbrev x23)
```

```
; Affected leaves are any that exhibit leaf spots, shreading, shot-holing or discoloration.  If leaves in different locations show
; different symptoms, chose the answer that indicates the position of the most severly affected leaves.

(condition_of_leaves_below_affected_leaves nom (does_not_apply unaffected general_yellowing defoliated nondescript) abbrev x24)

; Chose the answer that best describes the leaves below those leaves that are most severly affected.
; If the diseased leaves are scattered on the plant, or if the diseased leaves are near the lower nodes, chose "does not apply."

(leaf_withering_and_wilting nom  (does_not_apply absent present) abbrev x25)

; Be careful not to confuse withering and wilting with premature defoliation.  Leaves can wither and wilt without falling
; off or dying.  If the leaves are completely dead, chose "absent" as the answer to this question.

; ***************************** Stem descriptors *******************************

(condition_of_stem nom (normal abnormal) abbrev x26)

; The stem is abnormal if there are lesions or scabs (cankers) on it, if it is discolored, if it appears decayed internally or
; externally, if the stem is bent and the plant is paralell to the ground, if web-like growths are present on the stem, or if
; fruiting bodies (small growths protuding from the stem) are present.

(stem_lodging nom (does_not_apply absent present) abbrev x27)

; Lodging causes the stem to bend over, making the plant lie on the ground.
; Usually, this will occur for a whole area of a field, not just in isolated plants.

(stem_cankers
 nom (does_not_apply absent below_soil near_soil_line until_second_node above_second_node near_leaf_or_petiole_injury)
 abbrev x28)

; Stem cankers might be present in the from of a lesion (or scab) on the stem.
; Cankers may be isolated in blotches or spots, or may run extensively along lengths of the stem.
; Generally, cankers are of a different color than the natural color of soybean stem.  Cankers
; usually are not associated with decay or rot; if the area is decayed or rotted, it is probably not a canker.

(canker_lesion_color nom (does_not_apply none grey tan brown reddish_brown dark_brown_or_black) abbrev x29)

; Select "none" if the cankers are of the same color as the rest of the stem, but be very careful,
; canker lesion color is very indicative of the type of stem disease.

(reddish_canker_margin nom (does_not_apply absent present) abbrev x30)

; The margin should be clearly visible and be red or reddish brown

(fruiting_bodies_on_stem nom (does_not_apply absent present in_lines) abbrev x31)

; Fruiting bodies are abnormal growths which protrude out of the stem, aligned with its length.

(external_decay_of_stem nom (does_not_apply absent firm_and_dry watery_and_soft) abbrev x32)

; The signs of decay should be obvious if present.  If it is not very clear that the area is decayed,
; then this question should be answered "absent;" the problem area is probably a stem canker.
```

```
(mycelium_on_stem nom (does_not_apply absent present) abbrev x33)

; Mycelia are filamentous or web-like growths from a fungus.  They may extend along the length of the stem.

(external_stem_discoloration nom (does_not_apply absent reddish_brown brown silver_grey dark_brown black) abbrev x34)

; External discoloration is the color of the stem apart from any cankers or decayed areas.

(location_of_stem_discoloration nom (does_not_apply near_soil_line until_second_node above_second_node) abbrev x35)

; External discoloration concerns the parts of the stem that do not have
; cankers.  If any non-lesion area of the stem is discolored, indicate its location.

(internal_discoloration_of_stem nom (does_not_apply absent reddish_brown brown black) abbrev x36)

; You can determine if there is internal decay by breaking the stem in several places and looking for internal
; discoloration.  It is best to use a knife so as to get a good cross section.

(sclerotia_internal_or_external nom (does_not_apply absent present) abbrev x37)

; Sclerotia are hardened, thickened areas of tissue.  Try snapping the stem or squeezing the plant in a
; few places.  If the stem seems brittle or hard (not pliable) then sclerotia are present.

; ************************** Fruit Pod descriptors **************************

(condition_of_fruit_pods nom (normal abnormal) abbrev x38)

; Fruit pods are abnormal if they appear diseased, if they are spotted, if they are distorted, or if there are fewer than normal.

(fruit_pods nom (does_not_apply diseased few_or_none_present underdeveloped distorted) abbrev x39)

; If the fruit pods are not yet present, this question should be skipped.  Otherwise, examine the pods quite carefully; few diseases
; affect the pods in isolation, so this question can provide much useful diagnostic information.

(fruit_spots nom (does_not_apply absent colored_spots dark_blotches brown_spots brspots_black_specks) abbrev x40)

; Fruit spots are spots that occur on the surface of the fruit pods.

; ***************************** Seed descriptors *****************************

(seed_condition nom (normal abnormal) abbrev x41)

; This question refers to the seeds in the pods of the developing plant, not the seeds planted in the ground.
; This question is important only in the later stages of the season; if the seeds have not yet developed,
; answer "absent" or skip the question.  If the seeds have developed, they are abnormal if mold is growing on
; them, if they are discolored, if they are shriveled, or if they are smaller than normal.

(seed_mold_growth nom (does_not_apply absent present) abbrev x42)

; If the seeds in the pods appear rotted, or if any growth at all appears on them, then seed mold is present.

(seed_discoloration nom (does_not_apply absent present) abbrev x43)
```

```
; If the seeds appear discolored at all, or if there are spots of discoloration, the seed discoloration is present.

(seed_discoloration_color nom (does_not_apply grey reddish_brown brown black purple nondescript) abbrev x44)

; Open a seed pod and inspect the seeds.  If a spot occurs on any seed, try to characterise its color.

(seed_size nom (does_not_apply normal smaller_than_normal) abbrev x45)

; Normal seeds are ovular with a length of about 1/2" and a width of about 1/8".
; Anything smaller than this is smaller than normal.

(seed_shriveling nom (does_not_apply absent present) abbrev x46)

; Seed shriveling is present if the seeds in the fruit pods appear wrinkled or if they are distorted (i.e. not ovular).

;********************************* Root descriptors *************************

(condition_of_roots nom (normal abnormal) abbrev x47)

; Roots are abnormal if they are rotted, if galls or cysts appear on them, or if they are brittle.

(root_rot nom (does_not_apply absent present) abbrev x48)

; Root rot may appear as a decay on the stem base and older roots, as
; a sunken canker or as a discolored rot along the root and lower stem.

(root_galls_or_cysts nom (does_not_apply absent present) abbrev x49)

; Galls and cysts appear as small, white nodes along the roots and lower stem of the plant.
; The nodes are generally small, but large enough to be very noticeable.

(root_sclerotia nom (does_not_apply absent present) abbrev x50)

; If the roots are brittle or hardened (not pliable) then sclerotia are present.
)
;*****************************************************************************

classes class

variables
(class
 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x19 x20 x21 x22 x23 x24 x25
 x26 x27 x28 x29 x30 x31 x32 x33 x34 x35 x36 x37 x38 x39 x40 x41 x42 x43 x44 x45 x46 x47 x48 x49 x50)

events (by-ordinal-value
(0 6 2 2 3 1 0 1 0 2 6 1 1 1 1 2 3 2 1 1 2 1 1 3 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 2 4 1 1 0 0 0 0)
(0 5 2 2 3 0 1 1 0 2 6 1 1 1 1 2 3 2 1 1 2 1 1 3 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 2 4 1 1 0 0 0 0)
(0 6 2 2 3 1 1 1 0 2 6 1 1 1 1 2 3 2 1 1 2 1 1 3 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 2 4 1 1 0 0 0 0)
(0 5 2 2 3 0 0 1 0 2 6 1 1 1 1 2 3 2 1 1 2 1 1 3 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 2 3 1 1 0 0 0 0)
(0 5 2 2 3 1 0 1 0 2 6 1 1 1 1 2 3 2 1 1 2 1 1 3 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 0 0 0 0 0 0 0 0 0 0)
(0 6 2 2 3 0 1 1 0 2 6 1 1 1 1 5 3 2 1 1 2 1 1 3 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 0 0 0 0 0 0 0 0 0 0)
(0 5 2 2 1 3 0 1 0 2 6 1 1 1 1 5 3 2 1 1 2 1 1 3 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 0 0 0 0 0 0 0 0 0 0)
(1 6 2 1 3 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 5 4 1 2 1 2 1 0 1 1 1 1 4 1 2 1 1 2 2 0 0 0 0)
```

(1521101010000000000000000001154121210111140000000000)
(1521311010000000000000000001154121111011120000000000)
(1521101111000000000000000001154111110111240000000000)
(1521321010000000000000000001144121110111240000000000)
(1521311110000000000000000001144111110111241224220000)
(1521101010000000000000000001144121110111141224220000)
(2321111002342113221111223100000000000001211123220000)
(2211300102104211322111122110000000000000111000000000)
(2311110102331113211111221100000000000001211124220000)
(2412310102342113222111221100000000000001110000000000)
(2221301102104211322111122310000000000000121000000000)
(2420021102342113222111223100000000000001210000000000)
(2510130102642113221111221100000000000001110000000000)
(3311320002332223122121223100000000000000001210210100)
(3222113102332223122121224100000000000000121000000000)
(3310330102662213111111223100000000000000000000000000)
(3221122102662223122111224100000000000001211210210000)
(3321012102332223122121224100000000000000000000000000)
(3221302102332213122111224100000000000001210110000)
(3411330102332223122121223100000000000001210000000000)
(4222312102861213221111131111100111331100000000000000)
(4322322102861213221111131111540111331112100000000000)
(4222312102832213221111131111540111331111100000000000)
(4222312102862213211111131100000000000001110000000000)
(4122101102861213211111131100000000000000000000000000)
(4422221102631213211121133111540111331100000000000000)
(4322201102661213211111131111540111331100000000000000)
(5502311101000000011111531211100111103100000000000000)
(5301121110000000000000000001110011110210000000000000)
(5401311101000000011111131211100111002100000000000000)
(5401120102661211311121130111100111031000000000000000)
(5301311102611111311121110111100111223100000000000000)
(5411311102101121131111211102111001112231000000000000000)
(5502312102610121131111215302111001111031000000000000000)
(6602322010000000000000000001143122161421130000001112)
(6502311001000000011111232211100221614211300000012 12)
(6611320010000000000000000001146122142421130000001112)
(6511310010000000000000000001110022162421130000001112)
(6402121010000000000000000001146122141420000000012 12)
(6411311010000000000000000001110021110420000000001112)
(6502011010000000000000000001110022110421131124211112)
(7311311010000000000000000000000000000000000000000001121)
(7411301010000000000000000000000000000000000000000001121)
(7511311010000000000000000000000000000001211110211121)
(7511301010000000000000000000000000000001211110211121)
(7411311010000000000000000000000000000001211110211121)
(7411301010000000000000000000000000000001211110211121)
(7511302001000000011121231100000000000001211110211121)
(8821322110000000000000000001110031110111131210220000)
(8512321110000000000000000001110021110111131210220000)
(8522321110000000000000000001110031110111131223220000)
(8522322010000000000000000001110021110111131222220000)
(8622321110000000000000000001110021110111130000000000)

(85221211100000000000000000011100211101111300000000000)
(86221210100000000000000000011100311101111312232200000)
(96213311100000000000000000012542321331211212101100000)
(94213310100000000000000000012542222153120000000000000)
(94210320100000000000000000011342222121110000000000000)
(95213310100000000000000000011361211311200012101200000)
(93113311100000000000000000011342311311100000000000000)
(96113300100000000000000000011541211331200012222200000)
(95121300100000000000000000011542211231100000000000000)
(10321100101000000011113131100000000000000000000000000000)
(10521322102621213311113131100000000000000000012101100000)
(10420111101000000011113131100000000000000000000000000000)
(103223011021010121321111313110000000000000000000000000000)
(10221331101000000011113131100000000000000000000000000000)
(10521322102621113211223131100000000000000000012212100000)
(10521131101000000011223131100000000000000000012101100000)
(11522320102561213211111131100000000000000011200000000000)
(11421110102521213211111131100000000000000000000000000000)
(11622302102561213321121131211551111101111211241100000)
(11321330102711213111111131100000000000000000000000000000)
(11422121102561213211111131100000000000000000000000000000)
(11621310102561213111111131111551111101100000000000000)
(11522300102521213311111131200000000000000011200000000000)
(12322310102311211321211131100000000000000000000000000000)
(12422121102311211321121131100000000000000000000000000000)
(12222330102311211321111131100000000000000000000000000000)
(12112311102311211311211131100000000000000000000000000000)
(12212120102311211311111131100000000000000000000000000000)
(12212112102211211321121111100000000000000000000000000000)
(12221331102311211311111131100000000000000000000000000000)
(13311301010000000000000000011100131513100000000001211)
(13511301001000000011131611212361111523100000000001211)
(13211302001000000011131231211100131511100000000001211)
(13311300010000000000000000012100111521100000000001211)
(13222302010000000000000000011100131313100000000001211)
(13122301010000000000000000011100131513100000000001211)
(13311100010000000000000000011341131521112100000001211)
(14410310101000000011112131100000000000000000000000000)
(14210030101000000011112131100000000000000000000000000)
(14501101101000000011112131100000000000000000000000000)
(14301321101000000011112131100000000000000000000000000)
(14210310101000000011112131100000000000000000000000000)
(14101101101000000011112131100000000000000000000000000)
(14610030101000000011112131100000000000000000000000000)
(15621320102991213211111131100000000000011211251100000)
(15620310110000000000000000000000000000000011211251100000)
(15320320110000000000000000000000000000000011251100000)
(15622310101000000011111431100000000000011211251100000)
(15621310110000000000000000000000000000011211251100000)
(15620100101000000011111431100000000000011211251100000)
(15321320110000000000000000000000000000000011251100000)
(16120302010000000000000000012341122311100000000001211)
(16111101010000000000000000011341121311100000000001211)

(16 1 1 1 3 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 1 1 2 1 3 1 1 1 0 0 0 0 0 0 0 0 1 2 1 1)
(16 2 2 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 3 4 1 1 2 1 3 1 1 1 0 0 0 0 0 0 0 0 1 2 1 1)
(16 2 1 1 3 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 1 1 2 1 3 1 1 1 0 0 0 0 0 0 0 0 1 2 1 1)
(16 2 2 0 1 0 2 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 4 1 1 2 1 3 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0)
(16 1 2 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 3 4 1 1 2 1 3 1 1 1 0 0 0 0 0 0 0 0 1 2 1 1)))

C.8. Results of Experiments on Inductive Learning with Noisy Data

This sections contains the tabulated data for the results of the experiments with noisy data in Section 4.4. In the following tables, an attribute name is given as the trial name for single attribute noise in that attribute. *Average* indicates the average of all single attribute noise trials, *ALL* indicates a trial with noise in all attributes, and *CLASS* indicates a trial with noise in the classification information. Values are given for noise levels from 0 to 100 percent, in increments of 10 percent. In the first table, for noise in testing events only, each table entry is the average of 10 runs. In the remaining tables, each entry is the average over 5 runs.

NOISE IN TESTING EVENTS ONLY

PERCENT ERROR

Noise Level / Trial	0	10	20	30	40	50	60	70	80	90	100
maxilla_crown_spines	0.0	3.0	5.4	8.3	8.6	13.4	15.4	16.7	19.9	22.6	23.0
maxilla_lateral_setae	0.0	2.5	4.5	7.7	10.7	12.6	12.8	16.0	19.0	20.1	24.1
inner_canine_teeth	0.0	5.6	9.8	17.0	23.7	24.1	31.3	37.7	44.5	48.7	56.6
outer_canine_teeth	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
terga_mid_dorsal_pale_streaks	0.0	1.7	5.2	8.4	10.9	14.7	18.5	17.9	21.5	25.8	26.7
terga_dark_posterior_margins	0.0	4.0	6.3	10.8	14.5	17.5	20.9	24.2	27.4	30.2	32.2
dark_marks_sterna_9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AVERAGE	0.0	2.4	4.5	7.5	9.8	11.8	14.1	16.1	18.9	21.1	23.2
ALL (empirical)	0.0	16.7	29.8	39.2	50.1	59.8	69.8	73.1	77.2	82.1	87.0
ALL (computed)	0.0	15.7	27.6	42.6	52.6	59.6	67.3	72.9	79.5	83.6	87.4

NOISE IN BOTH TRAINING AND TESTING EVENTS
EXACT RULES (confidence = 1.0, utility = 0.0)

PERCENT ERROR

Noise Level / Trial	0	10	20	30	40	50	60	70	80	90	100
maxilla_crown_spines	0.0	1.5	4.0	4.5	4.4	2.8	3.8	1.7	1.4	1.0	1.7
maxilla_lateral_setae	0.0	3.3	5.1	6.1	6.0	7.7	9.4	13.6	12.5	13.6	14.8
inner_canine_teeth	0.0	2.6	3.6	2.1	3.3	0.9	2.0	0.0	0.0	0.0	0.5
outer_canine_teeth	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
terga_mid_dorsal_pale_streaks	0.0	0.4	0.8	0.0	0.0	0.8	0.0	0.0	0.0	0.0	0.0
terga_dark_posterior_margins	0.0	2.2	2.3	1.4	0.5	0.9	2.7	0.0	4.1	0.0	2.5
dark_marks_sterna_9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AVERAGE	0.0	1.4	2.3	2.0	2.0	1.9	2.6	2.2	2.6	2.1	2.8
ALL	0.0	14.5	30.8	49.2	57.5	64.1	73.1	78.2	83.8	83.9	86.3
CLASS	0.0	9.1	19.5	25.8	34.7	40.9	51.1	59.0	67.8	81.3	86.1

RULE COMPLEXITY

Noise Level / Trial	0	10	20	30	40	50	60	70	80	90	100
maxilla_crown_spines	4.7	5.1	6.2	6.7	6.5	6.5	6.4	6.0	6.4	6.2	6.0
maxilla_lateral_setae	4.7	5.1	5.5	7.0	6.8	7.6	7.2	7.6	9.1	8.0	8.3
inner_canine_teeth	4.7	5.6	5.6	5.9	5.9	5.9	6.0	5.9	5.9	5.9	5.9
outer_canine_teeth	4.7	4.7	4.7	4.7	4.7	4.7	4.7	4.7	4.7	4.7	4.7
terga_mid_dorsal_pale_streaks	4.7	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0
terga_dark_posterior_margins	4.7	5.1	5.4	5.3	5.3	5.3	5.4	5.0	5.0	5.0	5.0
dark_marks_sterna_9	4.7	4.7	4.7	4.7	4.7	4.7	4.7	4.7	4.7	4.7	4.7
AVERAGE	4.7	5.0	5.3	5.6	5.6	5.7	5.6	5.6	5.8	5.6	5.7
ALL	4.7	10.2	13.1	16.9	20.4	21.4	23.0	25.4	25.2	26.3	26.4
CLASS	4.7	11.9	16.9	19.2	21.5	22.9	26.4	26.7	28.6	30.3	29.7

CPU TIME (seconds)

Noise Level / Trial	0	10	20	30	40	50	60	70	80	90	100
maxilla_crown_spines	15	15	20	22	22	23	23	23	23	23	23
maxilla_lateral_setae	15	17	21	29	30	36	32	34	45	39	41
inner_canine_teeth	15	17	17	18	19	19	20	19	20	20	20
outer_canine_teeth	15	15	15	16	17	16	17	17	17	17	17
terga_mid_dorsal_pale_streaks	15	15	17	17	17	16	16	16	16	16	17
terga_dark_posterior_margins	15	16	19	19	18	19	17	18	18	18	18
dark_marks_sterna_9	15	14	15	16	15	15	15	15	15	15	16
AVERAGE	15	16	18	19	20	21	20	20	22	21	22
ALL	15	34	47	72	95	107	117	150	162	156	174
CLASS	15	41	72	89	111	129	156	167	165	190	174

NOISE IN BOTH TRAINING AND TESTING EVENTS
APPROXIMATE RULES (confidence = 0.9, utility = 0.1)

PERCENT ERROR

Noise Level / Trial	0	10	20	30	40	50	60	70	80	90	100
maxilla_crown_spines	0.7	2.6	3.9	3.9	5.9	4.1	3.8	5.1	5.9	4.3	3.0
maxilla_lateral_setae	0.7	2.5	5.5	5.5	6.7	8.1	10.2	11.1	11.3	11.2	11.6
inner_canine_teeth	0.7	4.2	3.4	4.0	5.9	3.1	2.9	2.4	2.8	2.8	2.5
outer_canine_teeth	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
terga_mid_dorsal_pale_streaks	0.7	2.1	0.7	0.7	0.7	1.1	0.7	0.7	2.4	1.8	0.7
terga_dark_posterior_margins	0.7	1.8	2.7	0.7	6.6	1.7	1.5	2.3	2.8	1.7	1.9
dark_marks_sterna_9	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
AVERAGE	0.7	2.1	2.5	2.3	3.9	2.8	2.9	3.3	3.8	3.3	3.0
ALL	0.7	17.4	31.6	46.1	56.1	67.0	73.4	77.9	81.4	85.9	84.5
CLASS	0.7	5.5	16.1	27.7	32.3	43.5	55.4	60.0	75.4	78.7	88.9

RULE COMPLEXITY

Trial \ Noise Level	0	10	20	30	40	50	60	70	80	90	100
maxilla_crown_spines	4.4	4.9	4.8	5.0	5.1	4.8	5.2	5.2	5.2	5.2	5.0
maxilla_lateral_setae	4.4	4.6	5.5	5.5	5.6	6.3	6.4	6.1	7.5	7.9	7.4
inner_canine_teeth	4.4	4.6	5.1	5.1	5.2	5.4	5.4	5.4	5.3	5.4	5.3
outer_canine_teeth	4.4	4.4	4.4	4.4	4.4	4.4	4.4	4.4	4.4	4.4	4.4
terga_mid_dorsal_pale_streaks	4.4	4.6	4.7	4.7	4.7	4.7	4.7	4.7	4.7	4.7	4.7
terga_dark_posterior_margins	4.4	4.7	4.8	4.7	5.0	5.3	5.2	5.2	5.2	5.2	5.3
dark_marks_sterna_9	4.4	4.4	4.4	4.4	4.4	4.4	4.4	4.4	4.4	4.4	4.4
AVERAGE	4.4	4.6	4.8	4.8	4.9	5.0	5.1	5.1	5.2	5.3	5.2
ALL	4.4	6.6	9.6	12.6	14.4	16.4	17.2	20.0	19.5	19.4	18.7
CLASS	4.4	7.2	11.2	13.1	16.6	17.9	19.1	20.1	20.0	20.6	19.2

CPU TIME (seconds)

Trial \ Noise Level	0	10	20	30	40	50	60	70	80	90	100
maxilla_crown_spines	15	17	17	20	22	22	22	22	21	22	22
maxilla_lateral_setae	15	16	20	23	24	30	31	30	41	40	39
inner_canine_teeth	15	16	17	17	18	18	18	18	18	19	19
outer_canine_teeth	15	16	16	16	17	17	17	17	17	17	17
terga_mid_dorsal_pale_streaks	15	16	17	17	16	17	17	17	17	17	17
terga_dark_posterior_margins	15	17	18	20	17	18	18	18	18	18	17
dark_marks_sterna_9	15	15	15	15	15	15	15	15	15	15	16
AVERAGE	15	16	17	18	18	19	20	19	21	21	21
ALL	15	27	43	62	79	97	105	126	139	145	137
CLASS	15	32	52	81	115	113	138	158	165	184	160

NOISE IN BOTH TRAINING AND TESTING EVENTS
APPROXIMATE RULES (confidence = 0.8, utility = 0.2)

PERCENT ERROR

Noise Level / Trial	0	10	20	30	40	50	60	70	80	90	100
maxilla_crown_spines	0.7	4.7	5.9	7.8	7.8	5.1	3.8	4.1	4.5	4.5	4.3
maxilla_lateral_setae	0.7	2.6	5.8	6.5	7.7	9.3	9.4	10.7	11.3	8.5	10.2
inner_canine_teeth	0.7	5.5	6.1	5.0	3.5	4.6	2.5	3.5	2.8	2.5	3.3
outer_canine_teeth	0.7	2.1	2.0	1.8	1.5	1.7	1.8	1.8	1.9	1.5	0.7
terga_mid_dorsal_pale_streaks	0.7	2.6	3.3	1.0	2.7	0.7	0.7	1.3	0.7	0.7	3.9
terga_dark_posterior_margins	0.7	2.1	2.8	2.9	2.8	2.9	1.7	3.8	4.0	2.1	2.1
dark_marks_sterna_9	0.7	1.0	1.3	0.7	0.7	1.3	0.7	0.7	1.0	0.7	2.9
average	0.7	2.9	3.9	3.7	3.8	3.6	2.9	3.7	3.7	2.9	3.9
ALL	0.7	21.4	38.1	44.8	61.2	65.1	76.4	81.3	82.5	84.5	85.6
CLASS	0.7	3.4	17.7	32.2	35.1	45.7	58.7	66.0	73.8	81.0	87.8

RULE COMPLEXITY

Noise Level / Trial	0	10	20	30	40	50	60	70	80	90	100
maxilla_crown_spines	4.7	4.8	4.8	5.0	4.5	5.1	5.5	4.7	5.5	4.8	5.5
maxilla_lateral_setae	4.7	4.9	4.8	5.3	5.1	5.6	5.5	5.2	5.4	5.0	5.3
inner_canine_teeth	4.7	4.9	4.9	5.2	5.3	5.4	5.3	5.3	5.3	5.3	5.4
outer_canine_teeth	4.7	4.1	4.4	4.3	4.3	4.3	4.2	4.5	4.5	4.3	4.7
terga_mid_dorsal_pale_streaks	4.7	4.6	4.9	4.9	5.0	5.0	5.0	4.8	5.0	5.0	4.9
terga_dark_posterior_margins	4.7	4.9	5.2	5.5	5.2	5.5	5.5	5.4	5.6	5.5	5.5
dark_marks_sterna_9	4.7	4.6	4.6	4.7	4.7	4.5	4.7	4.7	4.6	4.7	4.6
average	4.7	4.7	4.8	5.0	4.9	5.0	5.1	4.9	5.1	4.9	5.1
ALL	4.7	5.9	7.8	7.9	10.1	11.4	11.7	12.4	13.2	13.2	13.1
CLASS	4.7	4.9	6.6	8.8	9.1	9.0	9.5	9.1	9.2	9.1	10.5

CPU TIME (seconds)

Noise Level / Trial	0	10	20	30	40	50	60	70	80	90	100
maxilla_crown_spines	15	16	17	19	19	19	19	20	19	19	20
maxilla_lateral_setae	15	18	20	25	21	29	27	35	31	28	30
inner_canine_teeth	15	17	17	17	17	18	18	18	18	19	18
outer_canine_teeth	15	17	16	16	17	17	17	17	17	17	18
terga_mid_dorsal_pale_streaks	15	15	16	16	16	16	16	16	16	16	17
terga_dark_posterior_margins	15	16	17	18	18	17	18	18	17	17	17
dark_marks_sterna_9	15	15	15	16	15	17	15	15	16	15	15
average	15	16	17	18	18	19	19	20	19	19	19
ALL	15	27	36	43	68	77	93	108	108	125	130
CLASS	15	21	45	72	84	101	124	148	141	147	143

C.4. Fresh Water Distilling Plant Statistics

This table gives the percent error when testing all observed events against rules generated from a subset of the observed events. Run A was done with the confidence threshold at 1.0 and the utility threshold at 0.0 (exact rules). Run B was done with the confidence threshold at 0.9 and the utility threshold at 0.1 (approximate rules). Run C was done with the confidence threshold at 0.8 and the utility threshold at 0.2 (approximate rules).

PERFORMANCE IMPROVEMENT DATA

PERCENT ERROR

Run	Percent of Observed Training Events													
	0	3	5	7	10	20	30	40	50	60	70	80	90	100
A	85.7	60.8	54.5	40.7	39.4	25.4	15.8	13.5	7.2	6.7	3.8	0.7	0.6	0.0
B	85.7	60.8	54.5	40.7	39.4	23.0	21.0	14.8	15.3	12.5	9.7	5.2	12.1	6.8
C	85.7	60.8	54.9	41.4	40.6	27.5	24.1	22.5	25.0	23.1	20.1	14.8	19.0	13.5

C.5. Machine Readable Records

The program code and data sets used in this thesis are being maintained on a 1600 BPI Unix tar tape under the cognizance of the archivist of the Intelligent Systems Group at the University of Illinois Department of Computer Science.

Code files in directory ~jbecker/excel:

File	Description
cover.l	Bootstrap loader for the system
excel.l	Induction algorithms
excer.l	Deduction algorithms
backgrd.l	Background rule parser and applier
dataset.l	Data set management routines
sets.l	Generic set operations
dbvl.l	VL_1 data base operations
vl1.l	VL_1 selector and complex operations
parse.l	Data driven parser
arith.g	Grammar for background rules
textio.l	User interface

Code files in directory ~jbecker/simulate:

File	Description
start.l	Bootstrap loader for distilling plant simulator
components.l	Piping system component simulation code
network.l	Simulation network driver
runner.l	Critic, sensor, and effector routines for distilling plant

Data files in directory ~jbecker/excel/data:

File	Description
auto	Car-starting example data
chemistry	Bimetalic coordination compound data
mayfly	Stenonema mayfly description data
evaps	Freshwater distilling plant data

REFERENCES

[Machinist's Mate 3 & 2, 1972]
—, *Machinist's Mate 3 & 2*, Naval Training Command, NAVTRA 10524-D, United States Government Printing Office, Washington, D.C., 1972.

[Baim, 1984]
Baim, P.W., "Automated Acquisition of Decision Rules: The Problems of Attribute Construction and Selection," M.S. Thesis, Report No. UIUCDCS-F-84-922, Department of Computer Science, University of Illinois, 1984.

[Becker, 1983]
Becker, J.M., "AQINTERLISP: An INTERLISP Program for Inductive Generalisation of VL1 Event Sets," Report No. UIUCDCS-F-83-911, ISG 83-11, Department of Computer Science, University of Illinois, 1983.

[Becker, 1985a]
Becker, J.M., "Macros and Utilities for FRANZ LISP," Report No. UIUCDCS-F-85-937, ISG 85-7, Department of Computer Science, University of Illinois, 1985.

[Becker, 1985b]
Becker, J.M., "Topics in Incremental Learning of Discriminant Descriptions," Report No. UIUCDCS-F-85-935, ISG 85-5, Department of Computer Science, University of Illinois, 1985.

[Becker, 1985c]
Becker, J.M., "Word Sense Disambiguation Using Constraints and Supports," Report No. UIUCDCS-F-85-936, ISG 85-6, Department of Computer Science, University of Illinois, 1985.

[Copi, 1982]
Copi, Irving M., *Introduction to Logic*, Macmillan Publishing Co., Inc., New York, 1982.

[Cramm, 1983]
Cramm, S., "ESEL/2: A Program for Selecting the Most Representative Training Events for Inductive Learning," Report No. UIUCDCS-F-83-901, ISG 83-7, Department of Computer Science, University of Illinois, 1983.

[DeJong, 1983]
DeJong, G., "Acquiring Schemata Through Understanding and Generalising Plans," *International Joint Conference on Artificial Intelligence - 83*, pp. 462-464, Karlsruhe, West Germany, 1983.

[Falkenhainer, 1984]
Falkenhainer, B., "ABACUS, Adding Domain Constraints to Quantitative Scientific Discovery," Report No. UIUCDCS-F-84-927, ISG 84-7, Department of Computer Science, University of Illinois, 1984.

[Foderaro, Sklower, and Layer, 1983]
Foderaro, J.K., Sklower, K.L., and Layer, K., *The FRANZ LISP Manual*, Regents of the University of California, Berkeley, Califorinia, 1983.

[Hoff, Michalski, and Stepp, 1983]
Hoff, W., Michalski, R.S. and Stepp, R.E., "INDUCE-2: A Program for Learning Structural Descriptions from Examples," Report No. UIUCDCS-F-83-904, Department of Computer Science, University of Illinois, 1983.

[Hong and Michalski, 1985]
Hong, J, and Michalski, R.S., "Some NP-Complete Problems and an Approximate Method in the General Covering Theory," to appear as an ISG report, Department of Computer Science, University of Illinois, 1985.

[Hunt, Marin, and Stone, 1966]
Hunt, E.B., Marin, J., and Stone, P.J., *Experiments in Induction*, Academic Press, New York, NY, 1966.

[Lewis, 1974]
Lewis, P., "Taxonomy and Ecology of *Stenonema* Mayflies (Heptageniidae:Ephemeroptera)," Report No. EPA-670A-74-006, Environmental Protection Agency, 1974.

[Liu, 1977]
Liu, C.L., *Elements of Discrete Mathematics*, McGraw-Hill, Inc., New York, NY, 1977.

[Michalski, 1969]
Michalsi, R.S., "On the Quasi-Minimal Solution of the General Covering Problem", *Fifth International Symposium on Information Processing (FCIP 69)*, Vol. A3 (Switching Circuits), pp. 125–128, Yugoslavia, Bled, October 8–11, 1969.

[Michalski, 1974]
Michalski, R.S., "Variable-Valued Logic: System VL1", *Proceedings of the 1974 International Symposium on Multiple-Valued Logic*, West Virginia University, Morgantown, West Virginia, pp. 323–346, 1974.

[Michalski, 1977]
Michalski, R.S., "A System of Programs for Computer-Aided Induction: A Summary," *Fifth International Joint Conference on Artificial Intelligence*, pp. 319–321, 1977.

[Michalski, 1980]
"Inductive Learning as Rule-Guided Generalization and Conceptual Simplification of Symbolic Descriptions: Unifying Principles and a Methodology," *Workshop on Current Developments in Machine Learning*, Carnegie-Mellon University, Pittsburgh, July 16–18, 1980.

[Michalski, 1983]
Michalski, R.S., "A Theory and Methodology of Inductive Learning," *Machine Learning*, Michalski, R.S., Carbonell, J. and Mitchell, T. (Eds.), pp. 83–134, Tioga, Palo Alto, CA, 1983.

[Michalski and Chilausky, 1980]
Michalski, R.S. and Chilausky, R.L., "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*, Vol. 4, No. 2, pp. 125–160, 1980.

[Michalski, Davis, Bisht, and Sinclair, 1983]
Michalski, R.S., Davis, J.H., Bisht, V.S. and Sinclair, J.B., "A Computer-Based Advisory System for Diagnosing Soybean Diseases in Illinois," *Plant Disease*, April, 1983.

[Michalski and Larson, 1983]
Michalski, R.S. and Larson, J.B., "Incremental Generation of VL1 Hypotheses: The Underlying Methodology and the Description of Program AQ11," Report No. UIUCDCS-F-83-905, ISG 83-11, Department of Computer Science, University of Illinois, 1983.

[Michalski and McCormick, 1971]
Michalski, R.S., and McCormick, B.H., "Interval Generalization of Switching Theory," Report No. 442, ISG 1971-2, Department of Computer Science, University of Illinois, 1971.

[Michalski and Winston, 1985]
Michalski, R.S. and Winston, P.H., "Variable Precision Logic," M.I.T. AI Memo 95, to be released, Massachusetts Institute of Technology, 1985.

[Mitchell, Mahadevan, and Steinberg, 1985]
Mitchell, T.M., Mahadevan, S., and Steinberg, L.I., "LEAP: A Learning Apprentice for VLSI Design," Technical Report LCSR-TR-64, Computer Science Department, Rutgers University, 1985.

[Nilsson, 1980]
Nilsson, N.J., *Principles of Artificial Intelligence*, Tioga, Palo Alto, CA, 1980.

[Quinlan, 1983a]
Quinlan, J.R., "Learning Efficient Classification Procedures and their Application to Chess End Games," *Machine Learning*, Michalski, R.S., Carbonell, J. and Mitchell, T. (Eds.), pp. 463–481, Tioga, Palo Alto, CA, 1983.

[Quinlan, 1983b]
Quinlan, J.R., "Learning from Noisy Data," *Proceedings of the International Machine Learning Workshop*, Michalski, R.S., (Ed.), pp. 58–64, Department of Computer Science, University of Illinois, 1983.

[Reinke, 1983]
Reinke, R.E., "PLANT/ds: An Expert System for Diagnosing Soybean Diseases Common in Illinois. User's Guide and Program Description," Report No. UIUCDCS–F–83–911, Department of Computer Science, University of Illinois, 1983.

[Reinke, 1984]
Reinke, R.E., "Knowledge Acquisition and Refinement Tools for the ADVISE Meta–expert System", Report No. UIUCDCS–F–84–921, ISG 84–4, Department of Computer Science, University of Illinois, 1984.

[Rendell, 1983]
Rendell, L., "A new basis for state–space learning systems and a successful implementation," *Artificial Intelligence 20*, 4, pp. 369–392, 1983.

[Sammut, 1981]
Sammut, C.A., "Learning Concepts by Performing Experiments", Ph.D. Thesis, Department of Computer Science, University of New South Wales, 1981.

[Sammut and Banerji, 1983]
Sammut, C.A., and Banerji, R., "Hierarchical Memories: An Aid to Concept Learning," *Proceedings of the International Machine Learning Workshop*, Michalski, R.S., (Ed.), pp. 74–80, Department of Computer Science, University of Illinois, 1983.

[Simon, 1960]
Simon, Herbert, *The New Science of Management Decision*, Harper and Bros., New York, 1960.

[Spackman, 1983]
Spackman, K.A., "QUIN: Integration of Inferential Operators within a Relational Data Base," Report No. UIUCDCS–F–83–917, Department of Computer Science, University of Illinois, 1983.

[Stepp, 1984]
Stepp, Robert E., "Conjunctive Conceptual Clustering: A Methodology and Experimentation," Report No. UIUCDCS–R–84–1189, Department of Computer Science, University of Illinois, 1984.

[Winston, 1983]
Winston, P.H., "Learning by Augmenting Rules and Accumulating Censors", *Proceedings of the International Machine Learning Workshop*, Michalski, R.S., (Ed.), pp. 2–11, Department of Computer Science, University of Illinois, 1983.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-F-85-945	2.	3. Recipient's Accession No.
4. Title and Subtitle Inductive Learning of Decision Rules with Exceptions: Methodology and Experimentation			5. Report Date August 1985 6.
7. Author(s) Jeffrey M. Becker			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Department of Computer Science University of Illinois Urbana, IL 61801			10. Project/Task/Work Unit No. NSF DCR 84-06801 ONR N00014-82-K-0186 11. Contract/Grant No.
12. Sponsoring Organization Name and Address National Science Foundation, Washington, DC Office of Naval Research, Arlington, VA			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

This thesis describes a method of machine learning, embodied in a program called "ExceL", which allows the learning of decision rules with exceptions. Rules are learned by generalizing from specific examples. An exception to a rule is an example which does not agree with the rule. The degree to which exceptions are allowed is controlled by user established thresholds on the confidence and utility that the rules must have. Negative exceptions may be summarized in an "unless" condition.

The ExceL program is compared to one of its predecessors, an implementation of AQ, for differences in rule quality and efficiency. Experiments are done using noisy training data to see if there are limits on the performance of inductive inference under noisy conditions, and to compare exact and approximate rules under noisy conditions. Finally, a closed loop learning system which handles a simple control problem is demonstrated.

17. Key Words and Document Analysis. 17a. Descriptors

Machine Learning
Expert System
Inductive Inference
Learning from Examples

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 128
	20. Security Class (This Page) UNCLASSIFIED	22. Price

USCOMM-DC 40329-P71