# An Investigation into the Efficiency of Incremental Decision Tree Induction and Alternative Sources of Bias

A thesis submitted to the University of Manchester Institute of Science and Technology for the degree of PhD

1995

David M. Dutton

Department of Computation

# Abstract

This thesis presents an addition to the ID3 family of inductive machine learning algorithms. Our model, named JITTER, is based on ID5 and aims to eliminate any unnecessary work done whilst incrementally building decision trees. We show how it is possible to gather useful information from the training set and thereby augment tree manipulation procedures, effectively improving the efficiency of the induction process. Moreover, in certain cases it is possible to improve significantly the quality of the output. In addition, we illustrate how high levels of noise can greatly affect the efficiency of induction and how a straightforward approach can ameliorate these effects. Our algorithms use tree quality as a heuristic guide to reducing the number of entropy calculations, and the frequency and extent of attempted and actual tree revisions. The results show that our approach could prove useful to all decision tree algorithms, especially where speed, efficiency and parsimony are important, for example in Data Mining. We also offer a tentative framework of inductive efficiency and comment on its applicability to supervised induction in general.

# Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

# Acknowledgements

To my supervisor Gerard Conroy, thank you for your hard work, advice, suggestions and your conscientious effort on my behalf.

To my wife Claire especially, my family and friends, thank you for your encouragement, support and patience which have enabled me to get thus far. This is dedicated to you all.

To the numerous reviewers/examiners, anonymous and otherwise, of this magnum opus, thank you for your attention and constructive criticisms, your comments have undoubtedly improved this work, and my understanding. In particular I wish to acknowledge my father David (for translating this into English), Gerard Conroy, Nelly Khouzam and Andrew Bass.

To you all, I owe an enormous debt of gratitude.

Subvention from the EPSRC is gratefully acknowledged.

The patience and understanding of Dave Bloomfield and the BRE is also gratefully acknowledged.

# Contents

# Chapter 1

# Introduction

## 1.1 Artificial Intelligence

Artificial Intelligence (AI) is the field of study which aims to recreate, simulate and ultimately understand human intelligence through the use of computers. It is a somewhat eclectic discipline which has a number of overlapping sibling fields, such as Cognitive Science, Philosophy and Psychology, each offering different perspectives on AI's ultimate goal. Typically, researchers are interested in endowing a machine with the same characteristics which in a human would be considered indicative of intelligence. One such characteristic is *learning*, and it *must* be present for an autonomous agent to be capable of survival in a volatile environment. To put it simply, learning is the means of *acquiring* knowledge and we believe, is a fundamental process which underpins all intelligent behaviour. Indeed Langley (1987) holds that learning is present in every facet of intelligent behaviour and there could come a time when all intelligent 'systems' will include a learning component.

## 1.2 Learning

In a general sense, learning can be defined as the process of improving one's ability to do a task. For the main part of this thesis we refer to *cognitive acquisition* (such as learning to recognise an object by acquiring a description of it), rather than *motor skill refinement*, such as learning to ride a bicycle (e.g. Langley et al. 1987).

Michalski (1994) refers to learning as the combined processes of *inferencing* and *memorising*, where inference is deemed to be any kind of *reasoning* or *knowledge transformation*. That is, inference results in the production of new knowledge. Functionally speaking, learning will often consist of filtering data to obtain 'interesting' information and then refining this in some way so that 'useful' facets remain. Thus data is distilled into 'knowledge', whence it is stored for future use[1] .

## 1.3 Knowledge

What constitutes knowledge ? It is perhaps best described as rich information which will prove useful to the given system. It can be in the form of general rules-of-thumb, behavioural patterns, facts, figures, descriptions of objects and so on. We ignore more human values, e.g. emotions, and thus subscribe to *positivism*. Michalski (1994) holds that each piece of knowledge consists of its *content*, its *organisation* and its *certainty*, implying that knowledge is also structured and evaluable.

## 1.4 Machine Learning

Machine Learning is the name given to the study of learning using computers. The objective is two-fold: To learn about the spectrum of learning methods (e.g. human

[1] Our terminology is somewhat vague at present but suffices to characterise the processes we wish to discuss.

methods) and to endow computers with the same ability. Machine learning can therefore help one to understand *human* learning, through the construction of computer models, and perhaps discover *other* learning mechanisms which have not yet been recognised.

According to Shavlik et al. (1990) a system can learn either by acquiring new knowledge from an extrinsic source, or by modifying its own knowledge such that it becomes more effective. In the former case, learning entails the acquisition of knowledge without the act of directly programming it into a computer, and is directly relevant to this thesis. In the latter case, the type of learning is termed *speed-up learning* or *skill acquisition* (Shavlik et al. 1990), and is not dealt with here (see also Dietterich 1986).

Often the objective for a system is to acquire a description, or *intensional definition* of a given *target* concept. A *concept* can be thought of as a description of the common properties of a group of instances of the concept, or alternatively as a subset of a universe of objects. The latter would be termed an *extensional* definition. Thus a machine learning algorithm might be expected to construct a definition that is an approximation of some target concept (or goal), as indicated by a user. General overviews of the subject are given by Carbonell et al. (1983), Michalski (1983), Michalski et al. (1983b, 1986b), Rendell (1986), Michalski (1986), Kodratoff (1988), Carbonell (1989), Kodratoff et al. (1990), Shavlik et al. (1990), Kocabas (1991), Weiss et al. (1991), Michalski et al. (1994), which one should consult for more detail. Important early work is summarised in Dietterich et al. (1983) and includes work by Winston (Blocks World), Vere (THOTH), Hayes-Roth (SPROUTER) and Buchanan and Feigenbaum (DENDRAL). An excellent monograph is given by Angluin et al. (1983).

## 1.5 Applications

Acquired knowledge can fall into many categories, such as theoretical, methodological, factual, and technical (Kocabas 1991). Inductive algorithms (§2.3.5) can be used to detect patterns, trends and structure in many domains. For instance, it has been used to discover patterns in examples of chess end-games, without reference to the rules of chess (Quinlan 1983), which subsequently enable one to predict win or loss for a particular side. The knowledge acquisition stage of expert system development has been recognised as a potential bottleneck (e.g. Michalski 1983, 1986, Quinlan 1979), and any system which can automate the process, with minimal help from domain experts, is to be encouraged. Indeed, machine learning may actually improve on expert written knowledge (Michalski et al. 1980), and experts themselves (Cestnik et al. 1987, Kononenko et al. 1991). Similar techniques can be employed to refine knowledge-bases to improve completeness, consistency, simplicity and so on. For example, Shapiro (1987) uses *structured induction*, decomposing a problem into subproblems and using induction to solve each subproblem from the bottom up, building on previous solutions; Muggleton (1987) uses simplifying transformations to generalise data into a classificatory hierarchy; Michalski et al. (1983a) discover structure in a domain by clustering like objects together hierarchically. This structuring of a domain can provide valuable information and help to simplify complex concepts. Muggleton (1994) reports on the discovery of *new* knowledge, previously *unpublished.*

Other uses include fault diagnosis, condition monitoring, quality control and learning behaviour or properties (Donald 1994). More recently *learning apprentice* algorithms have been used to update an expert system's knowledge base as it operates (e.g. Bareiss et al. 1990). A similar approach is used by Winkelbauer et al. (1991), where different learning modules may be 'plugged into' an *Automatic Learning Environment.* Recent research

has seen the integration of several algorithms, together with knowledge representation and refinement methods into the "Machine Learning Toolbox" (Sleeman 1994). Indeed Sleeman appears to consider the merging of Knowledge-Based Systems and mainstream technology a priority, a concept with which we whole-heartedly agree. Multistrategy learning, where two or more disparate methods combine to augment each other, is presented in Michalski et al. (1994) for example. Other systems directly integrate learning elements within an expert system (Wang et al. 1991). Data Mining is a more recent development which entails the discovery of knowledge within databases (e.g. Holsheimer et al. 1994, Cai et al. 1991), and may well drive the need for more efficient learning. All should significantly improve future decision support systems.

## 1.6 Summary

A fundamental goal of AI is the succinct representation and efficient application of complex knowledge (Hinton 1990) and, as explicated above, machine learning entails the acquisition, organisation and refinement of such knowledge. In what follows, this will involve the learning of concept descriptions, based on input data from any given domain. It can occur within the context of differing levels of prior knowledge and possibly explicit targets or goals.

In a volatile environment, an autonomous agent's ability to learn quickly could prove to be necessary for survival. In a less dramatic situation, real-time knowledge-based systems should be capable of adapting to novel events. Thus we surmise that fast, efficient learning could become a prerequisite for this field's more extensive integration into more mainstream technology. In the sequel, we describe learning in these terms with regard to knowledge representation and algorithms.

## 1.7 Thesis Overview

The following chapter reviews major paradigms, algorithms, etc., initially from a general point of view, but progressively edging towards *decision tree induction.* We then review incremental approaches to decision tree induction (Chapter 3) and also criticise various incremental algorithms, with regard to the speed and efficiency with which they work. This chapter also includes some worst-case analysis of the algorithms involved, and we show how they can waste significant effort during learning. Chapter 4 represents the basis of our thesis and we proceed with descriptions of several extensions to the foregoing algorithms (see also Conroy et al. 1994, 1995). These are designed to manipulate the tree structures less, thereby doing fewer calculations and using less storage. We effect these increases in efficiency through the use of information obtained from the learning data, thus maintaining the general applicability of our algorithms. We empirically evaluate our algorithms (Chapter 5) and discuss the results in Chapter 6, with regard to the quality and efficiency of learning. Finally, our conclusions and further work are presented in Chapter 7. Appendix A explicates definitions and terminology, whilst Appendix B contains full results.

Occasionally we will delve into subsidiary subjects before our main theme of symbolic induction of decision trees. Suffice to say these are introduced briefly and will generally not be described further, they are included for completeness. We also favour a layout which briefly introduces each major paradigm, methodology etc. in an area, before a subsequent, separate section on the main topic of interest to us, wherein more detail is presented.

# Chapter 2

# A Review of Machine Learning

When designing an algorithm one must often make simplifying assumptions. From these, one will typically pick a method for encoding the necessary information (knowledge representation) and then go on to design the algorithm to effect the desired process. Qualities such as efficiency and effectiveness across many problem areas (e.g. Rendell et al. 1987) and simplicity, amongst others, are desirable.

A learning algorithm will typically be provided with *prior knowledge* to use during learning, a learning *goal* to guide execution and/or decide when a task is complete, possibly further *input* data, and methods for *transforming* the current knowledge (e.g. Michalski 1994). Below, we discuss some of these, and other considerations, starting with knowledge representations.

## 2.1 Knowledge Representation

Learning algorithms generally accept input in one 'language', or knowledge representation, and transform this into useful output, occasionally in a different language. The input language is only required to represent sufficient detail in the data being examined for the algorithm to undertake its allotted tasks. On the other hand, the output language

can often be required to represent more complex relations between constituent parts of the learned knowledge.

The formalism with which a system learns can have profound effects on the outcome of this process and should be adequate for the task in mind. If the input language cannot represent sufficient detail in the input data, one can rarely hope to learn useful knowledge. Similarly, if the output language is insufficiently expressive, one might never be able to find a suitable description for say, a group of objects. On the other hand, if it is too powerful, one might be able to generate many descriptions which are equally valid, but not be able to decide which is the most suitable depiction of the concept. Moreover, crucial features should be easily representable (Michalski 1983) and allow domain experts to comprehend and judge any knowledge acquired with ease. Dietterich et al. (1983) state that languages with a more expressive syntax provide greater precision but suffer increased complexity. Therefore one must seek to reach a compromise which is effective, often within a particular domain. Furthermore, a representation language can affect the generality of an algorithm, i.e. the task domains to which it is applicable.

### 2.1.1 Input Formalisms

Relatively few formalisms are used to represent the input to a typical learning system.

#### Feature Vectors

One method represents individual objects (cases, examples, instances etc.) of a concept as *feature vectors*. These are collections of prototypical features, or attributes, of the concept to be learned and are intended to describe its quintessential properties. They are grouped into vectors of attribute-value pairs, possibly with an associated class or category. Attributes can either represent domain 'primitives' (i.e. raw measurements), or more abstract features (Rendell 1986). Values can be numerical (ordinal) and dis-

crete or continuous, or categorical (nominal). The set of allowable relations between attributes and values is usually kept to '=', but can include others such as ' $\leq$' (Michalski 1983). These attributes and values describe an $A$-dimensional 'space' of possible example instantiations[1] .

Quinlan (1979) notes that properly chosen attributes can lead to a significant reduction in the amount of data an algorithm must handle and thereby a reduction in the search space (e.g. number of attributes). For example, Quinlan (1983) compresses a number of attributes into one *predicate* in a chess problem.

### Logic

A number of types of formal logic may be used, and can be considered to be an extension of the feature vector formalism. Feature vectors with classes can be considered to be simplified conjunctive implications in Propositional Logic. If one also allows disjunction a more powerful language is available.

### Numeric

Artificial neural networks also utilise feature vectors. However, due to the internal nature of the networks, these vectors are encoded in numeric form, e.g. as a string of bits indicating the presence or absence of particular attributes or attribute values. Continuous values may also be used. Genetic algorithms also tend to work with fixed length binary strings, representing the presence or absence of features.

### Other Input Representations

Some systems can accept a range of input types. For instance, an incremental system (§2.7.7) might accept a description in the *output* language in use (e.g. a tree), *and* new

[1] see Appendix A for terminology

input (e.g. individual examples), and produce further output (e.g. a new tree). Other systems (§2.3.3) accept problem solutions which depict a sequence of events planned to solve a problem. Thus the distinction we draw between the two types of knowledge representation is somewhat artificial as, theoretically speaking, an algorithm could accept input in any of the output forms discussed below.

### 2.1.2 Output Formalisms

#### Logic

It is possible to represent basic concept descriptions through the use of logical formulae and connectives. Thus if the presence of two properties together in an object signifies its membership of a particular concept, one could record this fact as $P \ \& \ Q \ \rightarrow \ R$. Formulae can be of arbitrary length, using the connectives & (and), $\vee$ (or), $\rightarrow$ (implies) and $\neg$ (not). Such systems can easily represent Boolean formulae in conjunctive normal form (CNF) and disjunctive normal form (DNF)[2] . It is much easier to learn *conjunctive* descriptions (Dietterich et al. 1983) because the expressiveness of the language is greatly circumscribed. Examples include Muggleton (1988) and Quinlan (1990a).

Extensions to this type of logic aim to improve the expressive power of the language and thus increase the space of representable concepts or descriptions. Such extensions include First Order Predicate Calculus (e.g. Vrain et al. 1988), but one might circumscribe the language and, say, learn simplified formulae, such as "Horn clauses". Others include Michalski's Annotated Predicate Calculus (APC) (1983) and variable-valued logics $VL_1$ and $VL_2$ (1980). Bain et al. (1987) use *non-monotonic logic* to aid incremental processing. *Fuzzy logic* (Zadeh 1994) adds degrees of 'imprecision' to predicates. Similarly, *uncertainty* means truth values are no longer Boolean but range across a real interval.

[2] See appendix A for definitions.

### Production Rules

Closely related to logic are *production rules.* Such condition-action rules typically consist of tests on salient features (attributes) of a description and draw a conclusion according to the observed values of those tests. The simplest form shadows propositional logic conditionals. Rules can be augmented with certainty factors which are probabilistic estimates of a rule's validity.

Rules are a concise way of representing a description, and are usually intended to be mutually exclusive. This can help to ensure *consistency* and *completeness.* Examples include Michalski et al. (1980, 1986a), Clark et al. (1987), and Laird et al. (1984).

### Decision Lists

Decision lists are a more structured form of 'if-then' rules, i.e. one can think of them as a list of "if-then-else" rules :-

IF condition-1 THEN conclusion-1,

ELSE IF condition-2 THEN conclusion-2,

ELSE IF .....

ELSE conclusion-n.

Rivest (1987) depicts each antecedent as a conjunct of $k$ terms, and each consequent in $\{true, false\}$. Such lists can be represented as trees (see below) which have two outcomes per internal node[3] . See (Shen 1992) for an incremental version.

### Decision Trees

A decision tree is a directed acyclic graph of nodes and arcs. Each node constitutes a test on some property (attribute) of a concept. Arcs emanating from such a node represent

[3] The right hand child (by convention) leads to a decision (a leaf) whilst the left-hand node leads to a subtree or leaf (e.g. GREEDY3 in Pagallo et al. 1989).

the possible outcomes of this test and typically lead to further nodes. Terminal nodes are termed leaves. Leaves generally contain an 'answer' for the 'question' asked of the tree. For instance, if a tree were used for classifying unlabelled objects, then a leaf would stipulate a class. A tree breaks a relatively complex decision into smaller subproblems, arranged in a hierarchy.

Trees can also be thought of as sets of rules. Each path from root to leaf represents one rule. Typically speaking, trees are an admirably simple method which have much intuitive appeal (e.g. clarity). As Quinlan notes (1979), decision trees can be isomorphic to programs and thus provide a very useful area of application for induction. However, decision trees constructed from labelled examples are not as descriptive as the types of logic described above (Arbab et al. 1988). Some systems use *binary* trees, i.e. trees which have at most two arcs emanating from any node, e.g. Breiman et al. (1984). Others are *V-ary* trees which can have up to $V$ child arcs[4] . Another representation formalism for trees can be seen in Cockett et al. (1989), where binary trees are represented *algebraically.* For a *binary encoding* technique for trees, see (Quinlan et al. 1989). A more general structure, where successive levels are positive and negative exceptions can be seen in (Bhandaru et al. 1991) and §3.1.5. Donoho et al. (1995) utilise AND-OR trees where branches are either AND'ed or OR'ed together and the leaves portray a concept description in DNF.

### Artificial Neural Networks

Artificial neural networks (ANNs, or neural nets) are fundamentally different to the machine learning aspects mentioned thus far. Typically the knowledge acquired is encoded in numeric form, as weights or coefficients within the network. Feed-forward nets consist of an acyclic directed graph, where nodes are simple processing units, known as *neurons.* Unlike trees, arcs (usually) connect nodes to all others in the next layer.

[4] $V$ is the maximum number of values per attribute

Network topology is typically fixed at the start of a run, unlike decision trees. Networks consist of a number of input neurons, zero or more *hidden layers* of neurons, and a final output layer, but see §2.2.2.

### Frames

Frames are a relatively rich repository for information. Each frame has a number of allotted *slots* which are filled in as learning progresses. Slots can be labels, conclusions, descriptions, actions to be performed, relations, and so on. Frames can be organised into a hierarchy and thus represent structure in a domain. Notions such as inheritance and subsumption are thus also applicable. Program fragments can be associated with a frame, and are often called *daemons*. These watch for special events or are 'fired' after, say, a slot update. Others dictate how a slot may be filled or updated. Examples include Blythe (1988), Lenat (1983), Núñez (1988), and Manago et al. (1991).

### Semantic Networks

Semantic networks (nets) consist of nodes which represent various entities (e.g. objects, classes, concepts) and are joined by arcs, representing relationships between nodes. Their inherent structure allows a hierarchy to be easily stipulated. Inheritance of properties from one level to the next is via *is-a* and *instance-of* links. Semantic nets have similarities with predicate logic, where terms are replaced by nodes and relations with directed labelled arcs.

### Other Formalisms

**FSA.** Angluin et al. (1983) mention finite state automata (FSA) that are 'acceptors', or parsers, of formal grammars. They embody the grammatical rules of a language in graphical form.

**Procedures.** Algorithms learn programs or procedures for accomplishing some task. Also known as automatic programming.

**Taxonomies.** An algorithm learns a structure within a domain and perhaps a range of specific-to-general concepts.

**Exemplars.** Exemplar-based learning is a relatively simple form of learning in which examples are stored in memory often in the form in which they are presented (see §2.5.7).

**Probabilistic Approaches.** Many of the above formalisms can be augmented with weights or probabilities to facilitate inexact reasoning. For example Fisher's COBWEB (1987a, b) and Schlimmer et al.'s STAGGER (1986a,b).

**Objects.** Object-oriented programming is a more recent technology (similar to frames) that enables procedures to be grouped with data into *classes*, enabling concept hierarchies, inheritance of properties, and so on.

### 2.1.3 Discussion

Wang et al. (1991) judge knowledge representations over four criteria (see also Ringland et al. 1988):-

1. expressiveness
2. ease of inference (loosely, *use*)
3. modifiability
4. extendibility

Detailed appraisal of these criteria is beyond the scope of this thesis, but we include some discussion for completeness. As noted above, an enlarged syntax increases expressiveness, but also computation (through complexity). On the other hand, greater expressiveness will often allow more compact representations and thus reduce computation through

reduced 'dimensionality' of the data. For a given input language, attributes which consist of categorical values have the effect of discretising the example space. One can think of this as taking larger steps through the concept space[5] .

Holte (1989) draws the distinction between a representation which is efficient with regard to the learning process but renders the *performance task* inefficient (or even intractable) and vice versa (e.g. exemplars and trees, but see later). It should now be apparent that optimisation of points (1) and (2) above is mutually exclusive and that they represent a trade-off to be satisfied, possibly per domain. Rendell (1986, 1987a, 1987b) makes use of a probabilistic representation for learning, and a simpler, more concise Boolean variation for subsequent transmission and/or explanation purposes (see also Kubat et al. 1991).

Modifiability implies ease of use across numerous domains. Many systems employ fixed and implicit knowledge, i.e. a designer has assumed a problem domain and an associated performance system, thereby providing necessary constraints on the learning process. As Keller notes (1987), this constitutes *compiled* knowledge which is correspondingly difficult to change. To paraphrase Marr (1982), any particular representation makes explicit certain information, at the expense of other information which is 'pushed into the background' and consequently may be harder to recover.

Other factors relevant to the choice of a representation might be its extendibility to a number of *levels* of knowledge, i.e. *meta-knowledge*. Such knowledge typically represents information about the *structure* of knowledge in a domain, or perhaps depicts a *strategy*.

Therefore we conclude that no single knowledge representation is generally inherently superior and that the choice of any one representation greatly depends upon the nature of the domain, the level of detail desired and the intended use of the knowledge.

[5] A *concept space* is roughly the set of all possible concept descriptions obtainable through the application of the algorithm's given operators. It is analogous to the *A-dimensional example space* mentioned previously but for the *output* language.

## 2.2 Levels of Abstraction for Learning

Learning can be modelled on a number of different levels of abstraction. For the purposes of this thesis, we subscribe to a simple taxonomy and think of learning as symbolic or sub-symbolic (numeric), according to the nature of the manipulations (reasoning/inference) which take place whilst learning. Dietterich (1986) concludes that there are two types of learning, *knowledge-level* learning, which equates to *knowledge acquisition*, and *symbol-level* learning which equates to *speed-up* learning (extant knowledge is used more efficiently). Kocabas (1991) also talks about learning at different levels, viz. the *knowledge*, *symbol* and *device* levels. These epithets are used in a different sense to Dietterich (above) in that Kocabas uses this trichotomy to classify algorithms according to their *predominant knowledge representation.* In some ways this is a little artificial as similar methods of induction are used, e.g. knowledge level learning is still, in our sense, symbolic. For instance, Rendell et al. (1987) draw a distinction between learning at the level of most inductive algorithms (e.g. examples and hypotheses), and at the 'higher' *meta-level* concerning knowledge *about* the learning task. We prefer to think of this as *symbolic* learning at different levels of *abstraction.*

### 2.2.1 Symbolic

*Symbolic* algorithms use arbitrary strings of characters to represent percepts of physical entities and other cognitive 'phenomena', such as relations. Thus symbolic learning uses 'pieces' of knowledge which have interpretable meaning, unlike numeric representations. *Reasoning* (knowledge transformation) is carried out at a largely *categorical* level unless probabilistic weights or *certainties* are used. Reasoning is therefore carried out at a higher level of abstraction than for numeric representations[6] . In this way, the output of learning

[6] In essence, one wishes to record salient information without unnecessary detail. Note that the level of *precision* changes, not the set of describable objects.

systems in the form of symbols is much more amenable to human comprehension.

### 2.2.2 Subsymbolic

Concepts are seldom naturally represented by such regimes above (e.g. Boolean) and, especially in the real world, are often only *partially* true. Subsymbolic systems are inherently numeric and typically represent descriptions across a collection of 'numbers'. For example with neural nets, a concept is represented by a monolithic network of neurons, synapses and weights. Genetic algorithms represent concepts across a population of numerically encoded individuals.

#### Neural Nets

Overviews of this topic are given in Rumelhart et al. (1986), Blum (1992) and Sethi (1991). As described above, a net consists of many interconnected neurons, one of which is shown in figure 2.1 (Blum 1992). In the figure, $A_j$ represents the activation levels

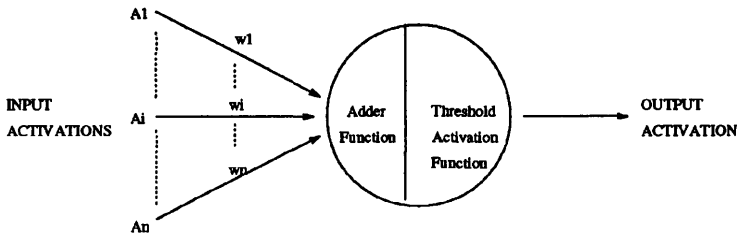



Figure 2.1: A neuron

from previous nodes. These are multiplied by the $n$ synaptic weights $w_j$ and summed at the neuron to ascertain the overall input level. Weights can be positive (representing excitory input in physiological terms) or negative (inhibitory). The input is subject to an *activation function* which decides the level of output from the neuron (e.g. $f = \frac{1}{1+e^{-ag}}$,

where $\alpha$ changes the curve's shape[7] ).

For each net, a topology, training algorithm and neuron model must be specified, including bias terms, weights etc.. During learning, networks may distribute a concept representation across many units, or may dedicate neurons to individual 'subtasks' (Hinton 1990, Carbonell 1989). Weight adjustment algorithms are many and varied but typically involve trying to minimise net error, e.g. mean squared error (MSE). A training method might use MSE to calculate an error after each example is passed through the net. A popular algorithm to update weights is the Backpropagation algorithm (BP) (Rumelhart et al. 1986).

For *supervised* learning, classified examples are propagated through the network. The net output is then compared against the desired classification. Unsupervised learning on the other hand uses unlabelled data and learns to cluster examples into classes.

Neural nets can take a long time to train and a net will select examples randomly from the set, with replacement, until an a priori level of accuracy or number of iterations is reached. Each run through the training set is termed an *epoch.* Noise and missing values can also complicate matters and hamper convergence.

### Genetic Algorithms

As the name suggests, Genetic Algorithms (GA's) are intended to correspond to the natural selection procedures observed in real life, and are basically an optimising search technique (Goldberg 1989). A GA starts with a 'population' which is ranked by an evaluation function. The best $K$ are chosen to form the next generation, and individuals are (possibly) combined with others by way of operators. These operators cause *crossover, mutation, reproduction* and so on, and thereby ensure that the 'genes' of the fittest

[7] This function represents a *soft* threshold, unlike the typically *hard* thresholds inherent in say, binary trees.

individuals are carried forward to the next generation. *Cross-over* entails the swapping of parts of the strings between two individuals, *mutation* randomly alters an attribute and *reproduction* copies whole strings. The new individuals replace the weakest ones in the current population and the process is repeated until a solution is found. The intention is that the top-ranked individuals will have the most favourable progeny.

For machine learning, the idea is to induce the best concept description, therefore individuals often start as examples of concepts and 'grow' into hypotheses in the concept-space. Fixed length binary strings represent attribute lists, and also concepts. Often a *don't care* value is also used, e.g. '#', and so one can form generalisations. Goldberg (1989) calls these *schemata*, which are templates describing subsets of strings. Goldberg introduces a simple GA based machine learning algorithm which consists of a rule and message system, a credit evaluation system and a GA. The system learns classificatory rules from the sequences of input 'messages'.

Spears et al. (1990) do not rate fixed length representations highly for general symbolic concepts and instead stipulate that their algorithm will learn fixed length rules. That is, the antecedents of rules will have a fixed number of attributes but each attribute has a variable number of possible values, represented by a bit string (one bit per *value*). Each population individual is a variable length *list of separate rules*. Fitness is judged by rule accuracy on a set of examples. An alternative approach would have each possible rule as an individual in the population, and this kind of approach can be seen in Quinlan (1988b).

Another use is demonstrated in Vafaie et al. (1994), where a GA is used to select feature subsets for use in discrimination rules. Wnek et al. (1994) use a GA to help learn rule antecedents and associated strengths[8] .

[8] Some authors prefer to think of GAs as symbolic, e.g. Carbonell 1989 and Kocabas 1991, but for the purposes of this thesis we concur with others, e.g. Wnek et al. 1994, Vafaie et al. 1994. Spears et al. (1990) state that GAs are general purpose and can be used in either setting.

### Statistics

We include statistical methods here due to their inherently numerical nature. One often requires to construct a probabilistic model to describe a 'sample', and subsequently draw conclusions from it, about a population in general. Estimating parameters of a given probability density function for a variable is one typical application, e.g. mean and variance (see Duda et al. 1973, Hawkins et al. 1982, Breiman et al. 1984). Statistical work which is of interest to the Machine Learning community mostly concerns *inductive inference*[9] , where one forms hypotheses based on data. Here, inference is based more directly upon probabilities within a given *sample* of observations.

Numerous techniques exist for investigating relationships between variables[10] . For example, *regression* attempts to determine the nature of a relationship between two variables (e.g. fit a curve to a scatter diagram of data).

*Bayesian techniques* describe decisions etc. in terms of *losses* or *risks*, e.g. one incorrect prediction may be more costly than another. To minimise overall risk, one chooses the action, or classification $c$, which minimises *conditional risk* (the risk associated with an action $a$, given an observation $e$). For this, one must use the Bayes rule which describes $p(c|e)$ in terms of a priori probability densities $p(e)$, $p(c)$, and $p(e|c)$[11] , e.g. Hoel (1984), Duda et al. (1973). A Bayes rule classifier is said to be optimal in that it depicts the minimum error one can expect in a domain.

*Cluster analysis* is described in Hawkins et al. (1982) and Duda et al. (1973), and covers many techniques used to sort and classify unlabelled examples into categories, possibly hierarchically, on the basis of 'similarity' measures. Clusters are groups of examples in $A$-space (see Appendix A) which have strong intra-class similarity. The 'distance' of

[9] See also §2.3.5.

[10] We restrict our discussion to two variables, but multivariate analysis is commonplace.

[11] This requirement is often a serious limitation, but one *can* estimate these probabilities from the finite sample.

an example from a cluster can be calculated by the sum of the squared distance for each attribute, from a cluster's centre. If two examples are less than some threshold distance from each other, they are grouped together. Clustering is basically an iterative optimisation problem to find the best partition, and is computationally intensive. Such clusters can then be used for classifying other objects, prediction of values, and so on. A similar approach can be seen in *nearest neighbour methods* (see also Weiss et al. 1991, Duda et al. 1973, and §2.5.7). Here the idea is to partition the example space into 'volumes' centred about an example in $A$-space and then let this volume grow to include $k$ nearest neighbours. A decision rule would then assign an example to the group nearest to it.

Statistical methods are often *parametric* in that they make assumptions about the data in use. For example, the form of the probability density function is given a priori, and even that it is 'normal'. Some parameters may be estimated from the current sample, often with good results, provided a sample is large enough for underlying assumptions to remain justifiable (e.g. see Fatti et al. 1982). Parametric methods therefore leave themselves open to criticism concerning the validity of assumptions and their generality.

### 2.2.3 Reinforcement Learning

Reinforcement learning is a paradigm where an 'episode' of temporally ordered states occurs. At each step the algorithm performs an action which is then evaluated for its quality and the algorithm receives a reinforcement, either 'reward' or 'punishment', according to the quality of the action (Cichosz 1995). Transition to the next state is then effected and the process reiterates (transitions and reinforcements may be stochastic), and the goal is to learn to predict a future outcome. *Temporal difference* learning, or *TD($\lambda$)*, aims to extract information from such sequences to predict future outcomes, and $\lambda$ is a factor which determines how many states are used, and their relative weight. For example, TD(0) uses the current state only (Dayan et al. 1994). Such techniques

are often used in 'control' to track the movement of an entity for instance, and may be implemented in a neural net. Dayan et al. (1994) prove the convergence of TD($\lambda$) algorithms generally but note that the speed of this convergence may be too slow in the 'real-world'. Traditionally, learning would wait until an 'outcome' is known, and having kept track of all predictions, calculate the difference between actual and predicted states to compute an error on which to 'train'. In TD($\lambda$), one does not wait for an outcome, and the 'error' between two states is used to speed learning (Cichosz 1995).

### 2.2.4 Hybrid Approaches

Attempts have been made to combine symbolic and subsymbolic algorithms, such as neural nets and decision trees (Sankar et al. 1991, Utgoff 1988b) and others (Towell et al. 1992, Sethi 1991). Utgoff et al. (1990) use *linear threshold units* (LTU) at each tree leaf. A LTU is a linear combination of (so far unused) attribute values, each with an associated weight. The weighted sum resulting from the input of an example is compared against a threshold. If the result is less than the threshold, the example is deemed to be negative, otherwise positive (the polynomial defines a hyper-plane §2.2.5).

Some algorithms select the number of neurons required as they train, e.g. (Sankar et al. 1991). It is possible to translate symbolic rules into neural nets (Towell et al. 1992), refine the nets and then retrieve symbolic rules (Towell et al. 1994), thereby overcoming some criticisms of neural nets (e.g. opacity and inability to justify solutions). Cios et al. (1992) say a decision tree corresponds to a hidden layer in a neural net. Sethi (1990, 1991) notes that a decision tree path represents an ANDing of tests, whilst several paths represent an ORing of terms. Sethi adds that neural nets perform similar operations across layers (§2.2.2) and so a tree can be converted into a net using transformation rules. Other *multistrategy* approaches may be seen in (Michalski et al. 1994), e.g. (Vafaie et al. 1994). One might also include statistical packages here, e.g. CART (Breiman et al.

1984) and AID (Hawkins et al. 1982), designed for statistical analysis, but adaptable to machine learning. It would appear too that Schlimmer et al.'s STAGGER (1986a) is a combination of symbolic and subsymbolic paradigms as it uses symbolic terms together with weights to describe a given concept.

### 2.2.5 Discussion

#### Linear Separability

Another way in which distinctions can be drawn between the different paradigms is with regard to the complexity of the decision regions produced. As we have described, an $A$-dimensional attribute vector establishes an $A$-dimensional *example space* of possible example instantiations. Within this space, classes of examples can often be grouped together into *decision regions* which form the basis of concept definitions. With real data especially, an algorithm's ability to delineate inter-class boundaries determines its ability to learn accurate and concise concept descriptions. In $A$-space, boundaries comprise *surfaces* of $A-1$ dimensions (a *hyper-plane* if the describing function is linear). The goal of any classifier then is to describe a boundary, or boundaries, within which, subsequent examples are deemed to belong to the appropriate class, and without, to belong to some other class(es). Problems are said to be *linearly separable* if the example space can be linearly dichotomised (i.e. a linear function describes a separating boundary). That is, in a two dimensional, two class problem, a straight line can be drawn between the two class regions.

One potential disadvantage with decision trees is their reliance on hyper-rectangular regions. That is to say, decision boundaries are orthogonal to the example space axes. For example, the linear function $x - y > 0$ is a 45° diagonal through the origin, and would lead to a 'staircase' of small rectangles in an ordinary decision tree formalism (see

also Weiss et al. 1991).

Other algorithms within the symbolic paradigm may use more complex decision functions, e.g. constructive induction (§2.7.6), Breiman et al. (1984), Murthy et al. (1994). In comparison, some neural nets *can* represent non-linear functions of attributes and thus decision regions in the concept-space are no longer restricted to hyper-rectangles.

### Comparisons

Each of the above formalisms offers advantages and disadvantages. In terms of the aspects we have discussed so far, neural nets can require thousands of iterations through the training data to converge to a description which is inherently opaque and therefore difficult to analyse and validate. It is often difficult to select an appropriate network topology and parameter values, requiring much experimentation.

Symbolic algorithms can produce descriptions in a fraction of the time, in a form which is more amenable to analysis and therefore validation. However, symbolic output can be somewhat crude (less accurate) and does not display graceful degradation in the presence of degrading input. In addition, neural nets are inherently parallel and thus specialised hardware can improve run-times, both for training and subsequent use. Hybrid algorithms attempt to overcome the faults of each formalism in a synergistic manner.

Numerous authors have compared the symbolic tree algorithm ID3 (§2.6.4) and BP (§2.2.2) as exemplars of each paradigm, e.g. Fisher et al. (1989). Fisher et al. conclude that ID3 is quick to learn and quick to reach a relatively high level of accuracy, whilst BP's learning curve is more gradual. Thus one might surmise that symbolic algorithms take bigger steps in the search space but may over- or under-step a solution.

Quinlan (1993b) describes neural nets, GAs and IBL (§2.5.7) as 'distributed' representations whilst decision trees and logic are not. The implication is that the former

types are less susceptible to small alterations, e.g. classification does not rely on one small part of a description. Hence distributed representations *may* be more robust in the presence of noise. Therefore one must conclude again that no one paradigm is inherently superior and each offers advantages in different domains. One's intended use must dictate the required method.

## 2.3 Symbolic Learning

There are a number of types of learning which, in a theoretical sense, differ in the amount of knowledge required and the level of inference performed. Obviously, the less information required to start the process the better, and the greater the level of inference, the less emphasis is placed on a user, or 'teacher', and the greater the level of inference, the more potentially complex and/or voluminous the derived knowledge becomes. Below we briefly introduce the main areas.

### 2.3.1 Rote

Rote learning consists of the pure acquisition of data. An algorithm records the data it acquires and possibly adjusts behaviour on the basis of this data. No filtering and/or refining of this data is undertaken and no judgements are made as to its usefulness. In terms of the amount of inferential effort required, it is at the bottom of our taxonomy.

### 2.3.2 Learning by Being Told

Learning by being told entails the acceptance of descriptions etc. from a teacher and adding this to one's current knowledge. In (Michalski et al. 1980), learning by being told occurs when an expert *provides* the knowledge to be utilised. It usually entails some transformation of the acquired knowledge into a form which can be easily related to prior

knowledge and thus some inference is performed (see also Kocabas 1991).

### 2.3.3 Deduction

Deduction is a basic logical "truth preserving" operation. Given a 'rule' such as $P \rightarrow Q$ and a fact $P$, then one can deduce that $Q$ is also true. Such rules form the basis of a *theory* (set of axioms, facts etc.). Herein lies the rub; one must have a theory with which to operate. Consequently, any theory must be applicable to the domain of application and ideally be complete and correct. Deductive theories are necessarily tailored towards a particular domain and the utility of a theory will diminish as the current domain progressively diverges from the intended domain.

This type of learning is also known as *analytical learning* (Carbonell 1989). It includes *macro-operator* creation (e.g. STRIPS in Fikes et al. 1971), *chunking* (Laird et al. 1984), *explanation-based learning* (ff.), and others (Kocabas 1991). *Macro* learning consists of learning aggregated operators, i.e. a sequence of steps which can be applied to speed up reasoning. *Explanation-based learning* consists of building a proof and modifying this to give a general explanation (e.g. Mitchell et al. 1990). *Chunks* are sections of search paths used in the solution of a problem, similar to the above macro-operators. These chunks are substituted for more low-level operator sequences when they are applicable[12] . This type of learning requires most inference so far, as one reasons from a current theory to deduce *new* knowledge.

#### Explanation-Based Learning

Many researchers hold that in order to learn complex concepts etc., one must start with a substantial amount of knowledge. This is intuitively appealing, if only from an observation of one's own experiences. In this paradigm, complex background knowledge,

[12] *Chunking* is used as a general mechanism for learning in SOAR (Laird et al. 1984).

or theory, is applied to few examples (typically one), to generate an *explanation* of why an example satisfies the definition of the concept in question. The general mechanism views an example with reference to the concept to be learned, and generates an explanation in terms of the extensive background knowledge. The relevant features of the example are identified, and the example can then be generalised, (this is termed explanation-based *generalisation* (EBG) e.g. Mitchell et al. 1990), to form the target concept definition. One can see that this type of learning involves both deduction (the explanation) and induction (concept generalisation). Explanation-based learning (EBL) requires (Mitchell et al. 1990, Kocabas 1991) a target concept definition, training example(s), a domain theory, and an operationality criterion. The operationality criterion is a description of the form required of the generalised concept definition. The task then is to find a generalisation of an example which forms a concept definition for the target, in terms of the operationality criterion (i.e. it is in a usable form). See also (Bareiss et al. 1990).

Minton (1988) uses EBL to learn search control knowledge, i.e. knowledge used to direct or constrain the search for a solution. Acquiring knowledge has an associated cost and it must be effective knowledge otherwise the cost will outweigh any gain it entails. Minton terms this the *utility* problem, i.e. there is a trade-off between search and knowledge.

Tadepalli (1989) notes that it may not be possible to *prove* that an example belongs to a target concept in some complex domains and thus one is confronted by the *intractable theory problem.* In such a case, *theory revision* may be possible (§2.5.8).

### 2.3.4 Analogy

*Analogical* and *case-based* reasoning (CBR), (Kolodner 1993, Watson et al. 1994) have many similarities and will be discussed together, in common with Sleeman (1994). CBR is reasoning based on previous experience adapted to new situations in order to solve

similar problems and can thus be seen to cover the whole reasoning cycle (Kolodner 1993). Processes include retrieving old solutions, adapting them to current purposes, integrating solutions into memory, and possibly general purpose problem-solving. New episodes are indexed (e.g. according to salient problem features) and stored for future use. Retrieval is based on similarity metrics, e.g. nearest neighbour matching, and may result in some elaboration or modification of certain solution steps. The more experience an algorithm has, the more applicable it is, and the more it is able to recognise successively smaller differences between cases. *Derivational analogy* can capture extra information during problem solving (e.g. justifications at decision points), which can provide constraints to adaptation during subsequent use (see also Kolodner 1993 and Watson et al. 1994). CBR can be seen to include both deductive and inductive learning (e.g. Michalski 1986, 1994, Kocabas 1991). See also Bareiss et al. (1990).

### 2.3.5 Induction

Induction is the process of *generalisation.* One observes the environment, e.g. objects within it, and then on the basis of this experience, forms cognitive models, rules, descriptions etc. which represent the observed relationships between objects. For example, one might use a description, or intensional definition, of a set of objects and thereby specify its *extension* (i.e. predict the class of unlabelled objects).

In logical terms, induction is said to be *falsity preserving.* That is to say, any rules generalised from a database of given facts might be true in the limited domain of reference but are not necessarily so in the 'universe'. In terms of our 'inference' taxonomy, induction resides near deduction.

### Learning As Search

For the purposes of this thesis, we characterise induction as "learning as search" (see Mitchell 1982, Michalski 1983, and Angluin et al. 1983). Michalski (1994) defines learning as a search through a *knowledge space*[13] as defined by the particular knowledge representation used. An output language describes a 'space' of possible concept descriptions, through which the learning system must search for an adequate representation. Laird et al. (1984) hypothesise that all complex behaviour can be characterised as search, and that this search is heuristic and goal-directed, i.e. there is some performance task in mind.

Many search procedures exist, for instance depth-first and breadth-first, with or without backtracking. However, these methods imply exhaustive searching, which is often intractable, hence the need for heuristics, e.g. *best-first* searching. One can characterise a search procedure in terms of *operators* for manipulating the current search state (one's working definition of the target concept). These operators define the possible steps one can take in the concept space, in terms of the output language and typically entail specialisation or generalisation of the current hypothesis. Thus concept descriptions are partially ordered by generality (Fisher 1987b). Michalski (1983) also uses a *reformulation* operator whilst Schlimmer et al. (1986b) may *invert* (negate) part of a description.

Langley et al. (1987) extol the virtues of *hill-climbing* as a plausible model for learning, where the search progresses under the auspices of an evaluation function. Thus a single description is adjusted as the concept-space search progresses until a suitably 'valuable' hypothesis is discovered. Gennari et al. (1989) describe this method as 'cheap' (in terms of memory), in that it avoids combinatorial explosions by maintaining only one current hypothesis, but that it is susceptible to example ordering, operator step size and local

[13] also *hypothesis* or *concept* space

minima.

### 2.3.6 Conclusion

The above paradigms can be used to draw up a taxonomy of learning algorithms. The level of inference undertaken in each has been described and is the basis of the said taxonomy. One can view an alternative taxonomy based on the amount of initial, or a priori knowledge built into an algorithm. Thus neural networks might be considered relatively knowledge-free, whilst EBL is knowledge-rich.

Induction works best when there are many examples, whilst EBL algorithms require very few. On the other hand, induction algorithms require relatively little background knowledge, or domain theory, and are thus easier to implement, easier to use and are potentially applicable to a wider range of domains. Sleeman (1994) suggests that in some complex domains, the definition of an extensive theory, or bias, might not be possible. Alternatively, see (Manago et al. 1991) for the use of complex background knowledge and deduction within a mainly inductive setting.

According to Kocabas (1991), 'similarity-based' techniques (e.g. ID3, §2.6.4) are limited by their inability to generate *justified* generalisations. Induction could be said to be justified by the examples used, (the more the better), but this is somewhat *shallow* knowledge and therefore cannot say *how* some concept is justifiable. EBL systems address this issue and to our mind, both techniques appear to be complementary. Effective learning in many domains could result from integration (see also Michalski et al. 1994, and cf. Bareiss et al. 1990). However, as intimated above, the requirement of an extensive domain theory can be a severe limitation, both to an EBL system's effectiveness and its range of applicability. EBL systems can suffer from incomplete domain theories, inconsistent theories and intractable theories (Kocabas 1991). A significant proportion of (Michalski et al. 1994) is given over to *theory revision* (see also §2.5.8).

## 2.4 Symbolic Induction

So far we have characterised the input and output representations of a learning system, the search for knowledge and some problems inherent in such an approach. We mention numerous prescriptive elements of (inductive) learning, and restate the main points :-

- The input language is sufficiently expressive to describe examples of the concept in sufficient detail.
- The output language is sufficiently expressive to represent the target concept.
- The examples are adequate exemplars of the target concept, i.e. the concept is learnable from these examples.
- Any background knowledge/domain theory/bias[14] employed is sufficiently general and correct not to preclude the target concept.
- The search operators allow sufficient movement through, and precision in, the concept space for the search to be tractable.
- A target concept definition is (possibly) provided.
- An evaluation function is (possibly) provided, to guide a search to the target description.

N.B. Exact definitions of 'suitable', 'adequate', etc. are not necessary at this stage, but will be defined below, as required. Our aim here is simply to adumbrate the main issues. Some algorithms learn from pre-classified data which is indicative of a given concept, whilst others are required to discover their own regularities in a concept space. Below we introduce various subfields of symbolic induction. Rendell (1986) gives an excellent, in-depth description of many topics discussed herein.

[14] see §2.5.2

### 2.4.1 Supervised Induction

*Supervised induction*, also known as *concept learning from examples, concept acquisition, similarity-based* and *empirical learning*, entails the use of a benign teacher, or domain expert. Such a teacher classifies examples into categories, such that they take on a pedagogical bent. A generalisation algorithm will then use these *training* examples to form its own concept definition. Other systems utilise an on-line oracle (e.g. Muggleton 1987). A simple method for supervised learning consists of using examples which are instances, or instantiations, of *only* the concept to be learned and are referred to as 'positive only' algorithms. Others utilise 'negative' examples, from some *other* concept(s), to constrain a generalisation (e.g. Mitchell 1982).

One can subdivide this area

again into algorithms which utilise inter- or intra-example relationships (Michalski 1986) to constrain generalisation. Algorithms such as ID3 (§2.6.4) use the similarities and differences between attribute values across all examples to differentiate between classes, an inter-example technique. On the other hand, intra-example techniques use one or more examples to constrain the applicable explanatory concepts (e.g. EBG, Mitchell et al. 1990).

### 2.4.2 Unsupervised Induction

In this case, (also known as *concept formation*) an algorithm is supplied with, or acquires, unclassified examples and is required to decide which examples belong to which classes, and indeed, what the classes are. Often little help from a teacher or environment is forthcoming regarding the number and type of concepts within an example space. Whilst 'supervised' descriptions might typically be used to recognise or predict class membership based on example properties, 'unsupervised' learning may result in descriptions which

specify the properties of classes of examples. This area can again be subdivided into *clustering* and *discovery* (Shavlik et al. 1990). Gennari et al. (1989) review concept formation in more depth.

### Conceptual Clustering

Michalski et al. (1983a) consider *taxonomic description*, or *conceptual clustering*. Clusters are groups of examples which have high intra-class similarity and low inter-class similarity. The problem then, is to cluster examples and then to characterise them, which gives rise to a two-phase search, firstly through a space of clusters, and then a space of concepts (Fisher 1987b). An algorithm attempts to classify training examples into such clusters and, in Michalski et al.'s case, arrange these into a hierarchy. Thus, a taxonomy is imposed upon a concept space and the clusters are the basis of concept definitions. One can think of concept learning from examples as a subproblem, i.e. with clustering, one groups examples into subsets, representing individual training sets for learning. Hierarchies often stipulate a generality ordering (decreasing downwards) of concept definitions, and thus inheritance of properties from one level to another is assumed (e.g. Langley et al. 1987). Note that in contrast, *partitions* are flat, i.e. non-hierarchical (Decaestecker 1991). It is usually desirable to minimise the number of clusters, i.e. one does not want one example per cluster. Clustering algorithms typically use a *distance* function to judge the difference, or similarity, between examples and/or clusters (see also §2.2.2 re. *statistics*). Fisher (1987b) briefly reviews similar systems.

Rendell et al. (1987) describe an incremental (§2.7.7) clustering algorithm, PLS1, which defines regions in the instance space from the top-down. One starts with a large 'hyper-rectangle' defined by all training instances and successively refines this into smaller regions on the basis of *dissimilarity* (similar to entropy, §2.7.1).

Fisher's COBWEB (1987a, b) constructs probabilistic, hierarchical concepts with

a view to improving *inferential ability.* This means the prediction of missing values whilst classifying subsequent examples. That is, classifying an object into a given class increases the amount of information one can deduce about it, because the class has a set of *normative* properties. (Normative attribute-values are statistically independent of others and thus highly predictive of a class.) COBWEB uses a statistical evaluation function known as *category utility* to find the classes inherent in the training examples. COBWEB uses nominal attributes whilst CLASSIT uses numerical values (Gennari et al. 1989), necessitating more complex 'utility' calculations. CLASSIT may handle noise better as it does not necessarily propagate all examples to leaves, and thus aims to avoid overfitting (§2.7.5). An augmented version of COBWEB, known as ECOBWEB is described by Thrun et al. (1991).

A similar approach is seen in UNIMEM (Lebowitz 1987, 1988), which can use both numeric and nominal attributes. UNIMEM also attaches 'certainty' type factors to descriptions for use in pruning and 'fixing', i.e. once a description exceeds a given threshold, it cannot be pruned, and if it should fall below another, it will be pruned away. UNIMEM also allows non-disjoint partitions allowing greater flexibility.

A similar approach is seen in ADECLU (Decaestecker 1989, 1991), where concept descriptions (one per node as above) consist of the collection of attribute values common to all examples in a node. As for COBWEB, each new example might cause further refinement of the concept at that node, via merge, split or create operators (*create* results in a new concept solely populated by the example). A subsequent version, ADECLU2 allows the selection of attribute value subsets in descriptions (COBWEB uses all).

Michalski et al. (1983a) note that most algorithms do not take the context into account, i.e. surrounding examples in the example space. They suggest the use of a concept-sensitive algorithm (CLUSTER/2) which also uses an externally defined set of concept definitions (i.e. the algorithm is supplied with a set of *allowed* clusters). Thus

the *conceptual cohesiveness* of two *points* depends on those points, the surrounding points and the set of concepts available to describe them.

### Learning By Discovery

This type of learning is also known as *learning from observation.* An algorithm observes its environment and induces 'concepts' by itself. Algorithms in this class can either simply observe the environment (passive), or they can *cause* changes in it and observe the effects (active). *Learning by experimentation* can be added here, e.g. LEX (Mitchell et al. 1983) generates problems in symbolic integration, solves them, and then analyses the solution for improvements (i.e. it learns heuristics in the form of production rules).

Schoenauer et al. (1990) describe a discovery system which performs operations on problem attributes reminiscent of constructive induction (§2.7.6). In the example given, the values of two attributes are found to vary in a similar fashion and so are combined into one, whereupon examples are redescribed in this extra attribute. Subsequent similarities allowed further accretion of this composite attribute with lone attributes to the extent that a mathematical law was 'rediscovered'.

Lenat's work on AM and EURISKO is included here (1982, 1983). AM gathered empirical data, 'noticed' regularities within it, and then defined new concepts which related the observations. It uses a large body of heuristics ("compiled hindsight"), provided as background knowledge, to recognise areas for investigation. These areas were noticed through the use of coincidence, extreme cases, the frequency of cases, and so on. Lenat uses heuristics to suggest plausible actions which enable a system to discover new concepts and to prune away implausible suggestions. This aspect is addressed in AM. As new knowledge is accrued however, new heuristics are needed, as their utility changes with the domain. This is addressed in EURISKO, where *meta-level* heuristics are used to discover heuristics. It was then noted (Lenat 1983) that new *knowledge representations*

were needed (recall that the usefulness of representations can change between domains), and were again developed through the use of heuristics (see also Donoho et al. 1995 and §2.5.8).

### 2.4.3 Conclusion

Supervised algorithms assume the presence of a benign 'teacher' so that examples are correctly classified. Inevitably this places a responsibility on a teacher to ensure that s/he is sufficiently expert in a domain. Due to the fallibility of humans, automation or avoidance of this stage is desirable. In an unsupervised setting this is at least partially avoided. On the other hand, induction from examples is not possible without constraining background knowledge (§2.5.2). One then concludes that the task for a concept formation algorithm (etc.) is even less constrained, especially if one requires a hierarchy of clusters. Computationally speaking, unsupervised learning is an intensive process, especially if one requires numerous runs to ascertain the best partition, and so the ubiquitous bias is often required in other forms. An example of this is a list of required concept classes to be used, as opposed to explicitly labelled examples (e.g. Michalski et al. 1983a). As before, such a list is required to be non-harmful (i.e. not misleading). Alternatively, fewer constraints can allow a search to unearth knowledge heretofore unconsidered, and even unknown.

The lack of a stipulated class can also allow queries about *any* concept attribute, not just an example's predicted label, as in supervised learning. Thus missing values are treated in a more intuitive and robust way. This method also allows more complex decision functions, i.e. polythetic versus monothetic classification (§2.7.2).

In conclusion then, one can consider supervised learning as a special case of the more general unsupervised paradigm. However, as learning in general is so difficult, one often requires a specialisation to ensure tractability.

## 2.5 Supervised Induction

Here we further explicate the field of supervised, symbolic induction. Within this paradigm one can include such methodologies as *Top-Down Induction of Decision Trees* (TDIDT), *Exemplar-* and *Rule-based Induction*, and so on. We begin by looking at supervised induction in a general sense and describe some relevant considerations.

### 2.5.1 Computational Learning Theory

Angluin et al. (1983) and Valiant (1984) have instigated much work on the theoretical aspects of learning, e.g. Valiant's *Probably Approximately Correct* (PAC) learning model. Others develop this topic further (Angluin et al. 1988, Natarajan 1991, Kearns 1990). These considerations are largely beyond the scope of this thesis, but some aspects offer a useful perspective on our work (Chapter 4).

Valiant (1984) is concerned with precise computational models of learning. It is desired to design learning algorithms which can learn classes of non-trivial concepts in a polynomial number of steps, and thereby characterise the classes of "learnable" concepts. The paper states that a class of concepts is learnable (e.g. $k$-CNF), with respect to the given learning protocol, if there exists an algorithm with the following properties:-

- run-time is described by a polynomial in: $h$ (an adjustable 'error rate' parameter), parameters specifying the size of the concept, and $A$ the number of attributes,
- for all concepts $f$ in the class of concepts $F$, and all probability distributions over the examples $e$, for which $f(e) = 1$, the algorithm learns, with probability at least $(1 - h^{-1})$, a concept $g \in F$ such that for all $e$, $g(e) = 1$ implies $f(e) = 1$, and summed over all examples, the probability that $f(e) = 1$ and $g(e) \neq 1$ is at most $h^{-1}$.

Kearns (1990) describes three areas in which the PAC model differs from empirical inductive learning. First, a learning algorithm is allowed to *approximate* a given target concept and is not required to identify it exactly. Secondly, the *computational efficiency* of learning procedures is now of central concern. Thirdly, one is concerned with *general* learning algorithms which perform 'satisfactorily', regardless of the probability distribution of the data. Kearns cites these aims as more realistic, as human learning is imperfect although rapid and general.

Natarajan (1991) describes learning tasks in terms of $\epsilon$ (the desired accuracy of a learned concept), $\delta$ (the desired confidence level) and $n$ (a maximum size in concept terms). For example (Natarajan 1991) if $\epsilon = 0.05$ and $\delta = 0.01$, then an algorithm will be 99% sure its output is 95% correct. According to Rivest (1987), as $\epsilon$ and $\delta$ approach zero, an algorithm will probably require more examples and time. In addition, once the required number of examples has been seen, any consistent concept should be sufficiently accurate, i.e. close to the target concept. The above parameters can be used to give estimates for the required size of data sets (i.e. the training set) and thus indicate whether a given concept is learnable. If the required example set size is too large, learning becomes *infeasible* for the given parameters (e.g. accuracy). A concept is generally held to be feasible if it is learnable in polynomial time, i.e. the algorithm's time complexity is polynomial in $E$, the number of input examples. However, such algorithms typically assume, that examples are not noisy, all values are present, and learning is not incremental (§2.7.7). However, Angluin et al. (1988) discuss coping with noise, albeit in a restricted from.

### 2.5.2 Bias

Although didactic training examples are supposed to impose constraints on the search for an effective hypothesis (Mitchell 1982), the hypothesis space is potentially huge and

as such can affect the tractability of a problem. The general argument is that there are too many concept descriptions consistent[15] with the training examples for an algorithm to consider in 'reasonable' time. Indeed, Michalski (1986) states that the number is theoretically infinite. Therefore one often requires a way to guide the search for 'likely' solutions. This 'bias' (alternatively known as *background knowledge, domain theory, heuristics*, etc.), is usually deliberately included in an induction system. Indeed, it has been noted (Michalski 1983, Rendell et al. 1987) that, computationally speaking, induction without bias is impossible, it is the sine qua non of induction. Thus one requires to shrink, or *prune* the search space. One should note then, that bias may affect the generality of an algorithm. Utgoff (1986) notes that a *strong* bias will focus an algorithm on a relatively small number of consistent hypotheses and a *correct* bias allows the algorithm to discover the intended target concept. Buntine (1990) decomposes bias further into *functional components*, viz. a *hypothesis space bias* (see below), an *ideal search bias* (which defines what an algorithm should search for), and an *application specific bias* (for the application, or purpose, in mind). An appropriate bias can facilitate 'inductive leaps' through the concept space (Mitchell 1982) and thus aid inductive efficiency. Bias is often intimately related to a learning algorithm's goals. Sleeman (1994) adds that the ultimate purpose or objective of the learning process should provide the most important bias for any system. Rendell (1987b) refers to *multi-layered* learning in which example, hypothesis, and bias *spaces* are integrated so as to exploit advantages offered by each.

Essentially, *bias* refers to the implicit and explicit, intentional and unintentional preferences imposed upon an induction algorithm by the designer and/or user. Rendell et al. (1987) describe bias as either *fixed, parameterised*, or *dynamic*. Fixed bias is set within an algorithm, whereas parameterised bias can be adjusted at the start of a learning session.

[15] A concept is said to be *consistent* with a set of examples if it would classify them into their correct classes (concepts) as indicated by an external oracle (teacher).

Dynamic bias is alterable during a run. Further flexibility is afforded by maintaining many biases, associated with different problems.

Bias can manifest itself in a number of guises, e.g. in the definition of the search algorithm (*procedural* bias) and in the definition of the hypothesis space (*declarative* bias). For example, with *language bias*, as previously described, the algorithm simply cannot express some possible descriptions because of a limited vocabulary. Another possibility is the set of *operations* allowed on a hypothesis at any one time. Alternatively, bias can be in the form of *heuristics*. As the search for an optimal tree, for instance, is generally held to be NP-complete (Hyafil et al. 1976, Fayyad et al. 1990), one must use heuristics to constrain the search. Wang et al. (1991) demonstrate the heuristic notion of a *shortcut*, which is used to learn search control knowledge to speed up future searches in similar situations. Examples include LEX (Mitchell et al. 1983), where the algorithm learns to associate operators with search states. Heuristics represent 'rules-of-thumb', or informal judgemental rules (e.g. Lenat 1983), *likely* to be of use. Typical heuristics used in learning might include greatest accuracy, least complexity, least cost (in training and execution), and so on. Often there are trade-offs to be considered, e.g. accuracy and complexity. Iba et al. (1988) make this relationship explicit to facilitate reasoning and thus decide which heuristic is preferable in which situations. In fact they argue that this same trade-off is inherent in all learning algorithms.

If bias is implicit then it is not revisable, whereas explicitly encoded bias can be, in effect allowing an algorithm to reason with it. In SOAR, Laird et al. (1984) include as much information as possible in an explicit, declarative sense. Utgoff (1986) details the algorithm STABB which will detect an *inappropriate* bias and shift to a 'better' one. Utgoff maintains that the increased search time expended in identifying a better bias can be justified if the current bias is inappropriate. Strong biases can be weakened by adding new terms to a language. Similar work can be seen in (Rendell et al. 1987)

where a variable bias management system (VBMS) is presented which also learns to associate biases with problem classes. This system will also alter the learning algorithm (e.g. parameters, operators) and evaluate an algorithm's performance in a domain (with respect to accuracy and primarily, speed), with a view to determining the best for the domain.

Michalski (1983) makes use of a *preference criterion*, i.e. a set of explicit bias terms altered by a user to influence the choice of consistent descriptions. Different domains require different preferences and thus Michalski defines a *Lexicographic Evaluation Functional* (LEF) which allows a user to specify the relative importance of the different criteria. The LEF is then used to rank competing hypotheses, starting with the first criterion in the list, allowing the 'best' hypothesis to be determined. Núñez (1991) augments decision trees with other background knowledge in the form of *is-a* hierarchies (a generalisation hierarchy connecting attributes) and attribute 'costs'.

### 2.5.3 Data-Driven Versus Model-Driven Learning

One can see that Mitchell's Candidate Elimination Algorithm (Mitchell 1977, 1982 and §2.6.1) is *data-driven*. That is to say, it takes examples and works directly with them to form consistent hypotheses. *Model-driven* algorithms depend upon a model (a representation of, in this case, learning) to guide induction. For example, ID3 (§2.6.4) uses *entropy*, which models learning as a search for maximum information in an example set. Model- or hypothesis-driven algorithms can handle a larger degree of unsuitable examples because they do not rely on the perspicuity of individual examples at each stage. However, a model-driven approach can result in a less focused search as it uses its model to generate new concept definitions. If a model proves to be inaccurate, an algorithm might consider many unwarranted hypotheses. Thus data-driven algorithms work bottom-up, generalising individual examples into sets, whilst model-driven algorithms work top-down, e.g.

by dividing example sets into subsets. Dietterich et al. (1983) conclude that top-down methods can be computationally more expensive, but are easily extended to learn other types of concept (e.g. disjunctive). Conversely, bottom-up methods may be quicker but less flexible. Rendell (1986) advises the use of both types, i.e. use of data-driven, or *evidential* criteria, and model, or *extra-evidential* criteria.

### 2.5.4 Concept Quality

One should be able to rate a concept description with respect to its quality, judge the relative merits of one description over another, and determine its suitability for its intended use. There are a number of dimensions over which one can judge quality and these include accuracy, clarity, efficiency, applicability and so on. Safavian et al. (1991) attempt to optimise *decision trees* with respect to error rate, path length, number of nodes and information gain. Searching for solutions which are 'best' globally can imply much processing, especially if no backtracking is possible (§2.7.7 ¶8) and so optimality is heuristically estimated at each step, i.e. there is limited, or no, look-ahead. Indeed, the generation of optimal decision trees is NP-complete[16] (Quinlan 1990c, Hyafil et al. 1976).

Cheng et al. (1988) suggest six measures for evaluating descriptions in *tree* form, including :-

- the number of tree nodes (indicates complexity),
- the error rate on unseen examples (indicates accuracy),
- the number of leaves in a tree (indicates how close a description is to the theoretical minimum of one leaf per class),

[16] For whatever dimension, e.g. size.

- the number of internal nodes in a tree (indicates the generality of a tree, i.e. if in rule form, the number of preconditions per rule),
- the average example support per leaf (indicates the applicability of individual tree paths, or rules),
- the number of decisions per test set example (indicates the efficiency of a tree [17] ).

Fayyad et al. (1990) perform a similar analysis of concept evaluation criteria and say that whilst accuracy is normally the most important criterion, its measurement can be time-consuming and error prone. They contend that minimising the number of leaves in a tree will bring an improvement in all other dimensions above (and that most of these are related). Probabilistically, reducing the number of leaves can be expected to reduce the error rate of a tree (Quinlan 1990c, Fayyad et al. 1990). This is in fact borne out by Murphy et al. (1994b) where error rate increased monotonically for *most* increasing tree sizes (except those near to the minimal size)[18] .

It is generally believed that a smaller tree will be more comprehensible (to experts) and will better generalise the training data (e.g. Bohanec et al. 1994). A popular heuristic known as Occam's Razor stipulates that the simplest explanation of an observed phenomenon is most likely to be the correct one. This is evidenced by the existing plethora of concept pruning techniques. Michalski (1983) introduces a *comprehensibility postulate* which suggests that inductive output should resemble that which would be produced by domain experts, both semantically and syntactically.

Accuracy is usually measured using the induced description to classify further examples. These examples can be taken from the training set, or from another, independent 'testing' set. The idea is to calculate an estimate of a concept's *true error rate.* As

[17] Note that this list has been simplified slightly to be in terms of trees instead of trees *and* rules.

[18] Note that this only applies to trees consistent with the training set.

concepts are inductive, unless one has all possible examples in the instance space, one can only ever estimate the true error rate. For an example $e$ of class $c$, and a decision function $f$ (which returns a class when given an example), the *true error rate* can be defined as $p(f(e) \neq c)$, i.e. the probability that the predicted class is wrong. Weiss et al. (1991) define true error rate as the error rate over an asymptotically large number of new examples which converge in the limit on the actual population distribution. Usually this will be estimated by $\frac{M}{E}$ where $M$ is the number of misclassifications in the test set, and $E$ is the size of the test set. A given concept description should be more accurate than the class probability distribution of its training set.

Various methods exist for determining accuracy. *Resubstitution* involves the use of the same training examples used to induce a concept. This can give an overly optimistic estimate of the true error rate as the description is tailored to the data. *Train-and-test* involves the subdivision of the data set into two random parts, one of which is used for training, the other for testing. Other methods include (Donald 1994, Breiman et al. 1984, Safavian et al. 1991):-

- random sampling
- repeated random sampling
- K-fold cross-validation
- 0.632 bootstrap

*Random sampling* equates to the above train-and-test, and can be repeated over $k$ random variations or runs to give an *average* error rate estimate. This is not ideal if data sets are small, as part must be left out for testing, thereby possibly leading to less accurate concepts. In *k-fold cross-validation* (Breiman et al. 1984, Safavian et al. 1991) the data set is randomly split into $k$ equal subsets. One set is then left as a testing set and a

concept is developed on the remainder. The test set gives an estimate of the concept's accuracy. This is repeated another $k-1$ times, omitting a different subset each time. The $k$ error rate estimates are then averaged. If $k$ is equal to the number of examples in the set, then this procedure is termed 'leave-one-out' and is useful for small example sets. Thus one uses the whole set for training *and* testing, but computation can be increased significantly. In *bootstrapping* (e.g. Donald 1994) examples are selected at random from the data set for training, with *replacement.* For a set of size $E$, this is repeated $E$ times so that the training set contains only a proportion of unique examples (on average 63.2%). Training and testing occurs as above (e.g. repeated $k$ times), and the error rate can be estimated with a simple formula.

Wnek et al. (1994) use *exact error rate* which is defined as *exact error* divided by the size of the example space. Exact error is the number of differences between the learned and the target concepts. Concepts are represented diagrammatically as groups of 'cells', where one such cell represents one position in the event space (one unique combination of all attributes and values). Errors can be errors of *omission* (false negatives, 'FN') where individual terms should be in the learned concept but are not, and errors of *commission* (false positives, 'FP') where terms should not be in the concept (Wnek et al. 1994, Weiss et al. 1991). Correct classifications are similarly labelled 'true positives' (TP) and 'true negatives' (TN), (Kononenko et al. 1991)[19] . An example use of this is for judging classifier sensitivity, e.g. $\frac{TP}{TP+FN}$. Indeed, Kononenko et al. analyse such facets of the 'gross' error rate estimate to more precisely determine performance. Other factors involved concern the use of prior class probabilities in calculating performance, i.e. the misclassification of an example of a more likely class is worse than for a less likely class. Kononenko et al. use an entropy-based calculation to more accurately gauge the

[19] assuming two classes

(probabilistic) amount of information contained in an 'answer'[20] .

Haussler (1986) uses two performance measures, viz. *convergence rate* and *computational efficiency.* He suggests that these dimensions are mutually exclusive and thus represent a trade-off. *Convergence rate* is defined as the size of the example set required to learn a hypothesis of a given error rate and confidence (probability). In (Wnek et al. 1994), it is defined as the number of examples seen before the concept prediction accuracy reaches 95%. *Computational efficiency* is the time taken to the final hypothesis. Concept or hypothesis *significance* is another method (but see §2.5.6 and Muggleton et al. 1992).

### 2.5.5 Characteristic and Discriminatory Concepts

Concepts may be said to be *characteristic* or *discriminatory.* Characteristic concepts are intended to capture *all* salient properties of a concept, whilst discriminant concepts are constructed in a way to differentiate between *given* concepts (Reinke et al. 1988). Characteristic descriptions are usually conjunctive and can become overly long in that they are intended to discriminate between all *possible* concepts. They are usually more appealing to humans but can become inaccurate (Carbonell 1989) because they do not necessarily discriminate effectively between the (subset of) example classes in a given situation. On the other hand, discriminatory descriptions include properties which will efficiently differentiate between given example classes, not necessarily including the 'normative' properties of a concept, and are usually predominantly disjunctive. One tends to prefer the shortest possible descriptions which will effectively discriminate between the *given* classes. Michalski (1983) says that characteristic descriptions are complete in that they cover all positive examples whilst discriminant descriptions are complete and consistent in that they also do not cover any negative examples.

[20] e.g. it is likely that a classifier returning an accuracy of 60% for a set $A$ (with the prior probability of the majority class of 25%), is better than for a set $B$ of 90% (prior probability 85%).

According to Kodratoff (1988), a robust learning system could easily make use of both types of concept, one to differentiate between a group of concepts and one to describe a concept once it has been recognised. Kodratoff states that for recognition (discrimination), a concept must be consistent, but not necessarily complete, and vice versa for description (characterisation).

### 2.5.6 Noise

When an example has an incorrect value for one or more of its constituent attributes, one refers to it as being 'noisy'. It generally implies that the particular attribute is inadequate for the learning and classification tasks. According to Weiss et al. (1991), any attribute which is no more predictive than chance can be considered to be noisy. However, it is noted that an attribute may appear noisy in isolation, but actually be highly predictive (of a class etc.) when used in conjunction with others. Noise can arise from a number of sources. In the real world particularly, tests which form the basis of example attributes can be wrongly administered, results can be incorrectly recorded, subjectively estimated, and data can simply become corrupted. Thus an example can no longer be discriminated from others of different classes which confuses the induction process. Mingers (1989) mentions *residual variation* where inadequate attributes mean that class-attribute relations cannot be 'properly' explained. Schaffer (1991) notes that attribute noise has a greater effect on an algorithm's performance than class noise. Natarajan (1991) describes *malicious noise* where whole examples are replaced. Similar problems occur when attribute values are missing, but see §2.7.4. Muggleton et al. (1992) use *compression* to identify noisy data. That is, a generalisation allows data to be 'compressed' into a hypothesis and the degree to which this is possible can be used to indicate its *significance.* Incompressible data is thus deemed to be noisy.

### 2.5.7 Major Paradigms

#### Inductive Logic Programming

Algorithms in the *ILP* paradigm learn Prolog clauses, from examples and background theory (clauses), e.g. Quinlan (1990a), Muggleton (1994), Cameron-Jones et al. (1994). Groups of these clauses then form *logic programs* (concept definitions). ILP has its roots in the CIGOL algorithm (Muggleton 1988, Muggleton et al. 1988, Bain et al. 1987) which uses constructive induction and an oracle to check the truth value of suggested generalisations. Muggleton (1994) holds that this formalism is more general than the 'classic' empirical learning (see below) due to the use of first-order relational logic, and also differs from EBL due to its use of (possibly) incomplete and/or incorrect background theory. Thus it can deal with structured objects, relations, $k$-ary predicates, and so on. Logical theorem-proving forms the basis of such program derivations and if formulae are constrained to Horn clauses, a Prolog interpreter can suffice. Such generality leads to large concept spaces and *extra-logical* constraints may be required to maintain efficiency. Statistical constraints may be used to rank hypotheses, whilst language bias restricts expressiveness e.g. by providing rule models or templates to describe clause form. A hypothesis $H$ cannot be *deduced* from background theory $B$ and positive example set $E^+$ (Muggleton 1994) but one can rewrite $B\&H \models E^+$ as $B\&\bar{E}^+ \models \bar{H}$ and then use (deductive) theorem-proving to get the negation of $H$.

Quinlan's FOIL (Quinlan 1990a, Cameron-Jones et al. 1994) is more closely related to ID3, i.e. top-down induction using a heuristic based on entropy. The emphasis is on an accurate, compact definition rather than an exact one. FOIL uses a 'covering' method (e.g. see §2.6.2) to learn Horn clause relations from examples of the relation. As long as a positive example remains and a logical clause to cover (satisfy) it can be found, the clause is added to the current definition and any positive examples covered are removed.

Finally the definition is pruned to remove redundant clauses. Quinlan makes use of the Minimum Description Length Principle (Quinlan et al. 1989) to infer *compact* definitions (§2.7.1), and a later version mFOIL, adds noise tolerance (Thrun et al. 1991). Muggleton et al. (1992) use hypothesis compressibility to identify and 'avoid' noise (§2.5.6).

### Exemplar/Nearest Neighbour Algorithms

These are also known as *Instance-Based Learning* (IBL) algorithms, e.g. Kibler et al. (1990), Aha et al. (1991), Aha (1992), Salzberg (1990), Quinlan (1993a), Bareiss et al. (1990), (see also §2.2.2 re. *nearest neighbour*, and cf. CBR in §2.3.4). An IBL algorithm consists of (Aha et al. 1991):-

- a *similarity function* which calculates the similarity between an example and the examples in a concept definition,
- a *classification function* which takes the similarity function's output and performance records of examples in the description and produces a class for the current example,
- a *concept description updater* which maintains performance records and decides which examples to include in a description.

Kibler et al. identify a number of models :-

- the proximity model, where all examples are stored, with no abstraction
- the best examples model, which assumes the existence of prototypes and thus only stores typical examples
- the selected examples model, which does not assume the existence of only one prototype, and thus may store more general examples.

Weiss et al. (1991) describe three types of distance measure (for calculating similarity), such as *absolute distance*, where the differences between the values of each attribute are summed. Others include *Euclidean distance* and *normalised distance*, that is, some attributes may be scaled differently, i.e. have different sized units, so one normalises all ranges to be within certain limits[21] .

Aha et al.'s algorithm IB1 (1991, and Aha 1992) is an incremental nearest-neighbour algorithm. IB2 saves on storage space by only 'remembering' *informative* examples. These can only be fully determined once the complete concept boundary is determined but one can approximate this with the set of misclassifications during learning (this assumes most misclassifications occur near the concept boundaries). IB3 adds some noise tolerance to IB2 by maintaining the number of correct and incorrect classifications, per example, and then performs significance tests to determine which examples are good classifiers and discards the noisy ones. IB4 (Aha 1992) adds weights to attributes to identify the relevant attributes over time, and are adjusted on the basis of classificatory performance (IB5 can also tolerate novel attributes).

Salzberg (1990) uses 'nested generalised exemplars', where some generalisation is allowed. A generalisation is a hyper-rectangle in the instance space. These hyper-rectangles can be nested within one another, and thus concept definitions can have exceptions which is an efficient, space saving addition. Exemplar-based learning therefore requires little domain knowledge (but see Bareiss et al. 1990). In Salzberg's case, each exemplar in memory has an associated probability that can be used in prediction, i.e. each exemplar has a record of its successes and failures as a predictor. Many exceptions lead to a fragmented concept space. Thus in order to cope with noise, exemplars are allowed to 'decay' over time, and will eventually disappear if not 'reinforced'.

[21] see Appendix A

### Rule-Based Algorithms

Rules are closely related to decision trees[22] , i.e. it is a simple task to reform trees as rules, but here we are interested in learning rules from the outset. For example, see Michalski (1980, 1983), Schoenauer et al. (1990), Shen et al. (1992), Wnek et al. (1994), §2.6.2 and ff.. For example, in CN2 (Clark et al. 1987, 1989), rules consist of conjunctions of attributes which predict class membership. CN2 generates rules one by one, by searching for a rule, adding it to the rule-base and then removing examples covered by (satisfying) it. The rule search entails looking for conjunctions of attributes that are satisfied by as many examples of *one* class as possible. The search starts with all possible most general rules, i.e. consisting of one attribute each. Each rule is evaluated for quality, statistical significance and the possibility of specialisation. In order to transform a description one can (Michalski 1980, 1983) :-

- drop conditions from a description (typically equates to removing a conjunct)
- add alternatives (introduce disjunction)
- extend references (enlarge the range of values of a term)
- close intervals (turn a list of say two possible values for a term into an inclusive range of values between them)
- climb generalisation trees (when terms are 'structured', replace a list of substructures with their parent structure)
- turn constants into variables
- turn conjuncts to disjuncts
- turn universal quantifiers ($\forall$) into existential ($\exists$)

[22] if based on Propositional logic

- use inductive resolution (as in theorem proving)
- use *extension against*, (for a given attribute, if disjoint subsets of its values appear in a positive and a negative example, use the subset appearing in the negative example to specify a range of values not allowed in the concept description)

### TDIDT

Decision trees are built in the following manner, assuming non-incremental processing (but see §2.7.7). The training set is scanned for a suitable test attribute which will form the root node. The values of this attribute are then used to classify the examples into (child) subsets. The process continues recursively on each child until all examples are in a solo-class leaf or until no further subdivision is possible[23] .

Quinlan (1993b) describes a number of prerequisites: Examples are described in attribute-value vectors, $A$ and $V$ are known a priori, there exists a predefined set of discrete classes, and 'sufficient' examples (but see §2.7.3). Trees can range from one leaf (i.e. a root node where all examples are of the same class) to a tree of many nodes and leaves. A maximally-sized tree (and consequently erroneous, see §2.7.5) will have one leaf per example and as many levels as there are attributes. The smallest, 'fully grown' tree will have one leaf per class. This paradigm will be described in more detail below (e.g. §2.7).

Binary trees are restricted to two children per test node. The CART algorithm (Breiman et al. 1984) searches each attribute in turn to determine the best (binary) partition of its *values*. It then finds the best split over the *attributes*. CART can use integer and continuous attributes because of its binary splits. That is, for an ordinal attribute '$a$', one has a split of the form $a < r$, i.e. a cut-point is determined and used

[23] In fact growing a tree until such pure leaves are attained is not strictly necessary and can be harmful, e.g. Cestnik et al. 1987, Quinlan 1986a, and ff..

as a threshold. Binary splits also allow more intelligent use of multi-valued attributes[24] . Utgoff (1995) notes the effect is to transform an attribute with a discrete set of values into a *propositional* variable that takes a value in {true, false}. It has been noted that $V$-ary trees may be converted to binary, with no loss of information (Sethi 1990, Utgoff 1995). Binary trees also have the advantage of normalising attribute informativity (§2.7.1). Weiss et al. (1991) say binary trees are often more accurate, but harder to construct. See (Mingers 1989) for a useful discussion.

### Discussion

Learning of rules will usually take place *bottom-up*, i.e. one starts with one or more examples, a very simple rule, and adds further clauses as more examples are processed. On the other hand, trees are often built in a top-down fashion, by scanning all available data and looking for a way to discriminate between all classes. The former approach can lead to inappropriate behaviour in the presence of noise, i.e. the algorithm is quite reliant on single examples (but on the other hand it may be less reliant on domain theory). The latter can cause problems when one is confronted with large amounts of complex data, as a typical algorithm would attempt to process all attributes of all examples (but see §2.7.7).

In GEM (Reinke et al. 1988) etc., a *selector* consists of an attribute and a subset of the possible values for it[25] , connected by a relation, such as '<'. Such a use of relations allows a more compact representation. For trees, a selector may be equivalent to a node test, and Breiman et al. (1984) state that binary trees generally allow 'smaller' descriptions to be learned (than $V$-ary trees). Decision trees are thus more restrictive than AQ for example (Michalski 1983, also Arbab et al. 1988). On the other hand, in

[24] Attributes with more than a small number of values.

[25] e.g. AQ15 (Wnek et al. 1994) allows internal disjunction and thus groups value into subsets in a rule, and so may be simpler than C4.5 rules

contrast to Michalski's STAR methodology, at each stage a decision is reached based on progressively smaller sets of the original data. With a STAR, one rechecks *all* negative and possibly many of the positive examples, at each generation of a 'cover'.

Safavian et al. (1991) note that trees can result in overlapping classes and therefore inefficient use of computing resources. That is, if two test nodes contain one or more common classes then the nodes are said to *overlap* and the same attribute tests may be replicated in separate subtrees (but see Van de Velde 1989, 1990). Núñez (1991) says that tree generalisation occurs through "dropping conditions" but extends this through the addition of "climbing the generalisation tree" and "adding alternatives" rules (via extra background knowledge). This extra flexibility may decrease bias, but may also hamper speed. KATE (Manago et al. 1991) is a variant of ID3 which uses frames and can thus cope with complex structured objects.

ILP also allows the use of relations and thus decision functions are potentially more complex than trees and some logical rules permit, i.e. one can learn $k$-ary relations. Muggleton notes (1994) that ILP systems are more general but are consequently under-constrained, e.g. the original FOIL implementation was limited to 128 constants and thus had difficulties learning some list relations (Quinlan 1990a). Thus simplifications and/or extra-logical constraints may be required. For example, Muggleton notes that FOIL requires non-numeric data and 'ground' background theory. It would also appear that ILP can suffer from recursive definitions, e.g.

```
ancestor(P,Q) :- ....., ancestor(P,Q), .....
```

(Cameron-Jones et al. 1994) obviously will not terminate on execution, and so extra checks may be required. FOIL avoids this by checking that recursive calls are not made with the same arguments, however, it cannot yet avoid indirect recursion.

Aha et al. (1991) stipulate that IBL algorithms are simple, which aids analysis and

more practical considerations such as implementation and testing (recall Occam's Razor). Aha et al. note that one can easily represent probabilistic concepts, but then it would appear that trees and rules may also do so, especially if these probabilities are based on the training set so far (i.e. record 'success' through misclassification count etc.). The authors also hold that IBL algorithms have relatively relaxed bias, as there is no explicit output language. This appears to be implicit in the concept boundaries and thus the similarity metric is of utmost importance. Indeed, similarity calculations are relatively straightforward for numeric data, but may be problematic for unordered attributes.

Bareiss et al. (1990) criticise algorithms' sole reliance on *higher-level* generalisations of examples. They state that the varied *uses* of concept descriptions necessitates more flexible representations which reflect inherent 'vagueness' (i.e. polymorphism in natural domains). That is, the use of inappropriate knowledge representations, into which a description has been 'compiled' may adversely affect the efficient use of this knowledge. Hence they propound exemplars and "lazy generalisation" which does less abstraction on fewer attributes, if examples are 'sufficiently' similar, and claim that this results in more generally applicable descriptions.

IBL assumes that similar examples have similar classes, and thus algorithms may be susceptible to noise. IBL algorithms avoid much effort in updating a description, but incur relatively large expense when classifying and may suffer from large memory requirements. Quinlan (1993) notes that IBL algorithms can lead to relatively complex and inaccurate concepts if irrelevant attributes exist (as compared to algorithms which determine explicit generalisations). Indeed, Quinlan combines IBL and 'model-based' methods (e.g. decision trees), and uses such models to adjust the $k$ prototypes selected for $k$-nearest neighbour matching (i.e. adjust the prediction made by a prototype).

The systematic comparison of many algorithms is presented in (Thrun et al. 1991) for the 'Monks Problems' which one should consult for further detail, and (Wnek et al.

1994), and others (e.g. Gams et al. 1987, Clark et al. 1989, Kocabas 1991). Rendell (1986, 1987a) describes and reviews logic, trees, and exemplars.

To conclude, one seems to be forever faced with a choice depending on an intended application, as again there is no one best algorithm. Each may 'score' better over some dimensions, such as simplicity, robustness, generality, efficiency, or accuracy, but often these dimensions are mutually exclusive, for example, more 'power' through an extended language syntax may decrease efficiency. On the other hand, adding bias (to improve efficiency) may decrease generality. One's aim then is accuracy, clarity and efficiency without bias, as will be explicated below. Each paradigm has much to offer and in future one may well see further integration of disparate algorithms in a synergistic manner.

### 2.5.8 Theory Revision

*Theory revision* is an area perhaps more commonly associated with EBL (§2.3.3) and/or ILP (§2.5.7), as both have explicitly represented background knowledge or domain theory. However, according to Donoho et al. (1995), theory revision "...integrates inductive learning and background knowledge by combining training examples with a coarse domain theory to produce a more accurate theory". Donoho et al. aim to change a theory's *structure* and/or its *representation* if either prove 'restrictive'. That is, a representation may be inappropriate for a new theory and may need to be changed (e.g. rules to neural nets, Towell et al. 1994), or a theory's structure may need more than 'local' changes.

## 2.6 A Review of Selected Algorithms

### 2.6.1 Candidate Elimination Algorithm

In Mitchell's Version Space approach (1982) one can see the *partial ordering* of the concept space through the use of a *more-specific-than* relation. A generalisation $G_1$ is more specific than another $G_2$, if the set of examples consistent with it is a subset of the examples consistent with $G_2$. This ordering is fundamental to Mitchell's Candidate Elimination Algorithm (CEA) (1977) where one starts with two non-empty sets $S$ and $G$. $S$ is the set of most specific concept descriptions (i.e. one of the actual examples per description) and $G$ is the set of most general descriptions, i.e. one that will 'accept' any example. Between these two states, one has the partially-ordered concept space. Thus moving from $S$ towards $G$, one generalises a concept description and vice versa. The intention is for a bi-directional search to converge on a single, consistent, mutually satisfactory hypothesis which then constitutes the learned concept. $S$ and $G$ summarise the previously seen examples. The use of disjunction complicates the search and noise may preclude convergence upon a description.

### 2.6.2 AQ/STAR

One can see the use of the *STAR* methodology in Michalski et al. (1980), Michalski (1983), Michalski (1994), and so on. A STAR is the set of maximally general expressions that are consistent with an example and do not 'cover' any counter-examples. Michalski notes that a given STAR may contain a prohibitive number of descriptions and therefore defines a *reduced STAR* that contains a fixed number of the best descriptions, as determined by an LEF (§2.5.2). A *cover* consists of conjunctions of attributes, disjunctively linked (Michalski et al. 1986a) and is used as the condition part of a rule. AQ builds

one rule per class by generating a cover (a disjunction of complexes). Each complex is a conjunction of selectors, which are attribute tests. The algorithm starts with one positive example and forms a STAR with respect to the negative examples (and any previously defined concept descriptions). The STAR is generalised as much as possible before the resulting descriptions are ranked. If the top description is not consistent with the remaining positive examples then the exceptions are used to start the process again until the description covers all positive examples and no negative examples. The resulting description is then reformed and contracted to remove superfluous terms. In INDUCE, the idea of a *partial STAR* is utilised, whereby some descriptions in a STAR may cover some negative examples (i.e. the strict consistency condition is relaxed).

A later algorithm, AQ15 (Michalski et al. 1986a, Kodratoff et al. 1990), is able to learn disjunctive descriptions in the presence of noise. Michalski et al. make use of an analogical matching function TRUNC, which enables a rule-base to be substantially simplified. The usual strict match is first used when testing a new example against the current hypothesis. If this fails, or it matches more than one class, then the algorithm attempts an inexact match, to find the best possible class. An analogical match is concerned with the measure of fit between attribute values of the new example and those in the concept description, taking prior class probabilities into consideration. A similar approach is seen in AQ17-FCLS which uses a two-tier concept representation consisting of a 'base' and other context dependent parts. AQ14-NT also uses TRUNC to implement noise tolerance, AQ17-HCI and AQ17-DCI perform hypothesis-driven and data-driven constructive induction respectively, and AQ15-GA uses a genetic algorithm (Vafaie et al. 1994). One should consult Thrun et al. (1991), and Wnek et al. (1994), for descriptions and comparisons of these and other versions.

### 2.6.3 CN2

Clark et al. (1987, 1989) describe CN2 as a cross between AQ and ID3, with added facilities to cope with noise. CN2 learns inexact production rules (consisting of conjunctions of attributes) in the presence of noisy examples, by relaxing the requirement that the rules produced be completely consistent with the example set (see also §2.5.7). The result is an ordered set of rules arranged in a *decision list* with a default rule on the end (§2.1.2. (Such an organisation removes the problem of conflict resolution during rule execution.) CN2 generates rules one by one, by searching for a rule, adding it to the rule-base and then removing examples covered by (satisfying) it. Clark et al. note that the rule search has a tendency to be very branchy and therefore requires pruning. There is a maximum number of 'rules-so-far' at any one stage, the weakest being removed after the tests on quality and significance. Rule (complex) quality is judged by an evaluation function that uses entropy, preferring rules that match as many examples of one class as possible and fewest of any other. Complex significance too is judged by an evaluation function designed to show that there is some correlation between attribute values and classes, and that relationships are not due to chance. The function used is:-

$$2\sum_{i=1}^{C} p_i log_2(\frac{p_i}{q_i})$$

where C is the number of classes, $p_i$ is the observed probability distribution of examples (over C) that satisfy the rule, and $q_i$ is the probability distribution of examples in the whole training set. Therefore one must also have a threshold below which rules are deemed to be insignificant. CN2 also includes a type of lookahead, used when the quality etc. of the fledgling rules is being assessed, to see if any specialisations of the rule in question *could* be 'significant'. If not, there is no need to actually generate them. CN2's significance test has the effect of stopping rule generation in areas of the search space where there are few examples. This can be considered a form of pre-pruning (§2.7.5).

### 2.6.4 ID3

In 1979 Quinlan introduced the ID3 algorithm. The original ID3 uses examples which have a fixed number of attributes, each with a small number of categorical values, and which are one of two classes (positive or negative). These examples are used to induce a decision tree which represents a classification rule for the target concept. The selection of attributes is controlled by an information theoretic measure *entropy*, in a *best-first* type search, with no backtracking. Whilst the trees are relatively efficient, they are not generally held to be optimal[26] , and construction of such a (binary) tree is NP-complete (Hyafil et al. 1976). That is, attribute tests are decided upon in a local sense as the tree is built and are therefore not considered to be globally optimal.

A *window* is employed whereby a subset of the available examples is chosen and used to produce a tree. This tree is then tested against the remaining examples and a limited number of randomly chosen misclassified examples are added to the window, and the process is repeated.

### 2.6.5 ID3 Descendants

ASSISTANT86 (Cestnik et al. 1987, Gams et al. 1987) and ASSISTANT *Professional* (Thrun et al. 1991) are able to cope with noisy, incomplete and inconsistent examples, and continuous and multi-valued attributes. ASSISTANT86 handles unknown values by assigning *all* known values to the attribute in question, together with a probability for each. Multi-valued attributes can make trees excessively wide and thus the binarisation of attributes is used (see also Breiman et al. 1984). If there are more than four values for an attribute, heuristics are used to choose the subdivision to avoid excessive searching. ASSISTANT86 can make use of both pre- and post-pruning. Three parameters are used

[26] e.g. minimise the number of tests required to classify an unlabelled example

to judge the reliability of nodes during training viz., *attribute suitability, class frequency,* and *node weight.* Suitability is a measure of the informativity of an attribute, i.e. its classificatory power. Class frequency specifies the proportion of examples belonging to the majority class at a node. Node weight measures the proportion of examples at a node with respect to the total number of examples. If too few are at any one leaf for a split to be judged reliable, splitting stops (see also Niblett 1987). Post-pruning involves the calculation and comparison of two error terms, *static* and *dynamic error.* If the dynamic error of a subtree exceeds its static error, the subtree is pruned.

Thrun et al. (1991) describe PRISM which uses a complex version of entropy. It recognises that some attributes and attribute values may be irrelevant to one or more classes in a problem, and learns rules directly, as opposed to trees.

C4 and C4.5 are described in (Quinlan 1990b) and (Quinlan 1993b) respectively. C4 grows a tree, adds some misclassifications to the 'window' as before and reiterates until no further improvement is forthcoming. The final tree may then be pruned and the *whole* process repeated with the most accurate tree overall being used. C4 uses the *gain ratio* (§2.7.1) to normalise attribute informativity and copes with continuous attributes and missing values and may also use a stopping criteria (§2.7.5). C4.5 is then able to generate a set of ordered, pruned rules from the final tree. Manago et al. (1991) use ID3 and frames in KATE.

### 2.6.6 CART

Breiman et al. (1984) introduce the CART algorithm which constructs binary trees. The authors use *node impurity* functions (Gini, Twoing, see §2.7.1) to decide when to split and which attribute to use. When there is no significant decrease in impurity, CART suspends splitting (pre-pruning), although Breiman et al. favour the post-pruning technique (§2.7.5). If an example space depicts a number of classes not easily separated

by orthogonal hyper-planes, the set of allowable splits at a node is extended to include linear combination splits of ordered attributes. This combination is given by: $\sum_m i_m a_m \leq v$, where $i_m$ are coefficients, $a_m$ are attributes, and $v$ is an attribute value. (See also Murthy et al. 1994.) One searches for an instantiation of this equation that minimises 'impurity'. The 'allowable split' set can also be extended with Boolean combinations such as $P\&Q \bigvee R$. However both can greatly increase computation as there are many such splits, and may cause tree comprehensibility to decrease. Classifying examples with missing values is handled by *surrogate splits*. If the best split at a node, on attribute $a_i$ is $s$, then find a very similar split $s'$ on an attribute other than $a_i$. This is repeated for the successive best surrogates at the node. An example with a value for $a_i$ missing, is then pushed to the left or right child according to the value for the surrogate attribute. If that is missing also, the second surrogate is used and so on. Gelfand et al. (1991) suggest a more efficient, iterative method for growing and pruning a tree. A set is divided into two equal parts; a *full* tree is trained on one and the second is used to pick the smallest, most accurate pruned subtree[27] . This second set is used to continue growing the tree, and the first set is then used to select a pruned subtree. This swapping process is re-iterated until tree size remains constant at the pruned stage (in a complex test, only four iterations were required).

### 2.6.7 STAGGER

Schlimmer et al. (1986a) introduce STAGGER where concepts are represented by weighted Boolean expressions. Each term of a definition is a set of attribute-value pairs connected by conjunction, and terms are disjunctively combined. Each attribute-value (AV) pair has two weights, representing logical sufficiency and logical necessity, which indicate positive and negative evidence for a class. STAGGER starts with the set of all unique AV

[27] picked from *all* pruned subtrees, not a parametric set as in CART.

pairs from the current example set (positive and negative only) and learns by incrementally adjusting weights and combining AV pairs conjunctively. STAGGER is able to use previously learned concept descriptions as new attributes in subsequent learning. It can cope with noise and concept drift through the use of the weights and will prune away ineffective description elements. Weights are adjusted through the success and failure of description elements in the prediction of example classes. STAGGER refines its concept descriptions through three operators, triggered by misclassifications. Specialisation occurs when AV pairs are conjunctively combined. Generalisation introduces disjuncts, whilst *inversion* negates 'poor' elements.

## 2.7 TDIDT

According to Safavian et al. (1991) there are four categories of methods for constructing heuristic decision trees[28] , namely bottom-up, top-down, hybrid and grow-prune. With bottom-up methods, pairs of examples are compared to compute a 'distance' which indicates that either the examples are sufficiently similar that they may be merged into one region, or conversely, they form two separate regions. One continues comparing and possibly merging regions until one arrives at the root, where one region remains, thus describing a tree structure. Top-down methods on the other hand rely on a splitting rule, which compares sets of examples, and partitions these into two or more subsets. Alternatively, Quinlan's $C4.5$ repeatedly grows and then prunes trees, and then picks the best one of the iterations (Quinlan 1993b). Here we focus on top-down induction and so one should consult Safavian et al. (1991) for further details.

[28] 'optimality' can only tractably be judged by using heuristics, i.e. truly optimal decision trees are NP-complete

### 2.7.1 Attribute Selection

One requires to split a set of examples into subsets which are 'purer' with respect to the relative concentrations of classes. There are a number of methods for defining *impurity functions*[29] and Fayyad et al. (1992) list desirable properties :-

1. maximise inter-class separation
2. minimise intra-class separation
3. be sensitive to permutations in class distributions
4. be smooth (differentiable) and symmetric with respect to classes

Keeping examples of one and only one class together will minimise the number of leaves and so provide more support for each[30] . Permutations of class distribution vectors indicate a change in the dominant class and that the attribute in question is a good choice for a split. *Symmetry* ensures no bias towards one particular class, *smoothness* implies the function 'behaves' between its maxima and minima and gives similar values for similar 'situations' (i.e. it is continuous). The use of *intra*-class separation is also seen in Quinlan's *model trees* (1993a).

#### Entropy

When initially considering a training set, the amount of information one has concerning the classification of examples is minimal. Tree building aims to maximise the gain in information, or 'certainty', at each stage (attribute selection). Entropy is non-parametric and is calculated thus (Quinlan 1979, 1983):-

$$\rho = -\sum_{i=0}^{C} p_i log_2 p_i$$

[29] One can use accuracy but it is flawed (Breiman et al. 1984).

[30] Fayyad et al. (1990) state that the number of leaves and the error rate are linked

where $p_i$ is the *a priori* probability of class $i$. This probability is approximated by the relative frequencies of classes at a node. A weighted sum of the information available at the children[31] is subtracted from the information available at the current node. The prospective attribute split which maximises the *gain* from parent to child is selected. The information gain from level $d$ to $d+1$ can be given by :-

$$-\sum_{j=1}^{C}\frac{c_j}{\sum_{k=1}^{C}c_k}log_2\left(\frac{c_j}{\sum_{k=1}^{C}c_k}\right)+\sum_{i=1}^{v}\frac{\sum_{k=1}^{C}c_{ki}}{\sum_{k=1}^{C}c_k}\left(\sum_{j=1}^{C}\frac{c_{ji}}{\sum_{k=1}^{C}c_{ki}}log_2\left(\frac{c_{ji}}{\sum_{k=1}^{C}c_{ki}}\right)\right)$$

where $C$ is the number of classes, $v$ is the number of values of the attribute currently being considered for the split, $c_x$ is the example count in class $x$, and $c_{xy}$ is the example count for class $x$ with value $y$.

As the process is recursive, and therefore operates on subsets at successively lower nodes, each attribute selection is therefore only locally optimised with respect to the selection measure. The bias inherent in ID3 is to prefer smaller trees, in which the paths are as short as possible. However, as Van de Velde (1989, 1990) notes, ID3 does not necessarily find the smallest decision trees consistent with the data.

Quinlan (1988a) considers the issue of multi-valued attributes. The original ID3 was found to prefer attributes with more values, ultimately resulting in larger trees. Quinlan's approach is to *normalise* the information shown by each attribute by using the *gain ratio.* The information gain available by using attribute $a$ to split a node is divided by :-

$$-\sum_{i=1}^{V}\frac{E_i^+ + E_i^-}{E^+ + E^-}log_2\left(\frac{E_i^+ + E_i^-}{E^+ + E^-}\right)$$

where $E^+$ is the number of positive examples at the node and $E_i^+$ is the number for value $i$ (out of $V$ values).

[31] i.e. given a potential 'split'

### AMIG

A related procedure is named Average Mutual Information Gain (Sethi 1990, Sethi et al. 1982). For a two class problem and one attribute with values $v_1$ and $v_2$, one maximises the following:-

$$\sum_{i=1}^{2}\sum_{j=1}^{2} p(c_i, v_j) log_2(p(c_i|v_j)|p(c_i))$$

### CART

The CART algorithm (§2.6.6) uses two impurity functions. The *Gini diversity index* is calculated for a node $n$, and $C$ classes (Crawford 1989) thus:-

$$\gamma(t) = \sum_{i=1}^{C}\sum_{j=1}^{C} p(j|t)p(i|t), i \neq j$$

One should note that Gini is similar to entropy but has no log, and is therefore simpler to calculate. The *Twoing criterion* is designed to give *strategic* splits, such that similar classes are grouped together near the top of a tree, and then near the bottom of the tree they are isolated. At each node, one separates $C$ classes into two supersets, and labels all examples with a class in group one as *class 1*, similarly for group 2. One then computes impurity as if it were a two-class problem and selects the conglomerate superclass pair which gives the greatest reduction in impurity, by maximising :-

$$\frac{e_L e_R}{4}\left[\sum_j |p(j|n_L - p(j|n_R)|\right]^2$$

for the two children $n_L, n_R$ of node $n$, with example proportions $e_L$ and $e_R$.

### Others

Quinlan et al. (1989) introduce the Minimum Description Length Principle (MDLP) as applied to the construction of decision trees. Trees are encoded in bit form with a view to minimising the 'cost', i.e. the description length of both the tree and any misclassified

examples. It is as if one wished to communicate this data in the shortest possible message. The method characterises the inherent trade-off between tree accuracy and tree size, i.e. one wants the smallest tree but also the most accurate. The MDLP is used in place of entropy to select attribute tests and subsequently to prune away subtrees. A fully grown tree is repeatedly pruned back, i.e. whenever the removal of a node improves total communication cost for tree plus misclassifications. Variations and others are given in (Murthy et al. 1994), Núñez (1991), and so on. Mingers (1989) gives an excellent survey of this area, including the use of $\chi^2$ for attribute selection, and Lopez de Mantaras (1991) uses a 'distance-based' selection metric. Ben-David (1995) criticises 'entropy-based' trees for "non-monotonic" behaviour with respect to classification, for domains with ordered classes (e.g. credit ratings). Non-monotonic trees, for example, may grant a high credit rating to an applicant when another, more suitable one is refused. Ben-David adds a weighted item to the usual entropy calculations such that a measure of "ambiguity" is included, which is designed to minimise non-monotonicity. *Ambiguity* is defined as the log of the ratio of the number of actual non-monotonic branch pairs, to the total possible number of such pairs. A branch pair is a node test plus a leaf class (e.g. in a tree path with three attributes $A_1, A_2, A_3$, and class $+$, one gets three pairs: $\{A_1, +\}, \{A_2, +\}, \{A_3, +\}$), and these pairs are checked against all others for non-monotonic behaviour.

### 2.7.2 Decision Functions Revisited

Fisher has described attribute selection in trees as *monothetic*, as only one attribute is considered at a time. *Polythetic* classifiers on the other hand, can consider groups of attributes for a given decision (e.g. Fisher 1987a, b, Schlimmer et al. 1986b). Typical decision tree algorithms look for relationships between one attribute and a class, i.e. they assume that attributes are independent of one another (Quinlan 1988b, Cios et al. 1992, Hawkins 1982).

ID3 suffers from limited lookahead in that it selects the 'best' attribute for splitting, according to entropy. ID3 could improve this situation by performing an $k$-ply lookahead, and thus maximise information gain over $k$ attribute tests (levels of a tree). This would be of benefit in "j-of-k" problems (e.g. parity), where a single attribute is useless for classification. Extensive lookahead is computationally infeasible in all but the smallest of attribute sets. ID5 (§3.1.2), for example, rectifies this situation to a certain extent by adding (quasi-) backtracking which can 'undo' suboptimal choices. IDX (Norton 1989) is based on ID3 but uses limited lookahead, as dictated at run-time, to select tests in each tree level. Trees become more balanced and average tree depth is consistently reduced. IDX also took note of individual test costs and is able to delay or ignore the more expensive ones.

Other approaches use combinations of attributes at a node, such as neural nets and constructive induction (§2.7.6). Utgoff et al. (1990) use LTUs (linear threshold units) to learn multivariate decision functions at tree nodes. LTUs consist of the untested attributes and weights, and training moves the hyperplane they represent about the example space, attempting to dichotomise the example classes. Such a hyperplane is of any orientation. Murthy et al. (1994) use *oblique trees* where a hyperplane is 'situated' at each tree test node.

Cheng et al. (1988) and Fayyad et al. (1992) present a generalised version of ID3 named GID3. Some values emanating from a node may be irrelevant to classification (identified statistically) and so these are grouped into a single 'default' branch. A similar algorithm is seen in PRISM (Thrun et al. 1991) and C4.5 (Quinlan 1993b), which recognise that some attribute-value pairs might be highly relevant, and some not so. As Murthy et al. (1994) note, the use of such decision functions can significantly decrease tree size, but they may become more opaque in the process.

### 2.7.3 Training Sets

Decision trees are generally *non-parametric*, i.e. they make no assumptions about the particular type of *distribution* of the underlying data. However, one normally assumes that the training set is representative of the 'population', or example space (Mingers 1989). Hypotheses approximating a target concept describe contiguous regions of an example space. A training set must have one or more example from each disjunctive concept region and must contain enough positive and negative examples to delineate the boundaries of these regions. The more disjuncts there are, the more training examples are needed to obtain an accurate concept description. An exhaustive, deterministic training set is not usually available and could easily be beyond the capability of many systems through sheer size[32] . Quinlan (1983) gives an estimated lower-bound on the accuracy of a tree in terms of the number of examples seen. If the number of examples seen is 'sufficiently large', the tree is predicted to be over 99% correct, independent of the size of the example universe.

It has been noted that the order of examples can affect the concept that is produced by an algorithm. Generally, one uses several randomly selected variations of the training and testing set pairs to build and test concepts, and then average the results to try to eradicate such anomalies. Back-tracking may also help undo poor choices.

Learning from 'near misses' represents an increase in efficiency (e.g. Schlimmer et al. 1986c, Utgoff 1988a, Gemello et al. 1989 ), i.e. the algorithm is forced to focus on areas where further information is needed. Near misses are typically counter-examples that are 'accepted' by the current hypothesis and help to avoid over-generalisation by ensuring consistency (and vice versa for rejected positive examples).

[32] Note that one would then be able to say that the induced concept was truth preserving because one had encountered all possible example instantiations.

### 2.7.4 Attribute Types

For continuous attributes, one typically selects a cut-point to 'split' on (for a binary split). For example, in C4.5 (Quinlan 1993b) a value set is ordered and a split midway between each pair is evaluated. Alternatively, one can discretise into subranges (e.g. Clark et al. 1989). Attribute values close to such a threshold are often better treated slightly differently (Quinlan 1987a, 1990b), as small changes in this value can cause radical changes in the outcome (i.e. class)[33] . One can specify *subsidiary cut-points*, $r^-$ and $r^+$, either side of a threshold $r$ (Quinlan 1987a). For a given $r$, an estimated standard deviation of the number of errors is calculated, and $r^+$ and $r^-$ are then chosen such that if $r = r^+$ or $r = r-$, the extra errors encountered are one standard deviation above $r$. One can then provide a probabilistic estimate of an example's class, according to the particular range within which a value ($v$) occurs ($v < r^-, r^- \leq v < r, r \leq v < r^+, v > r^+$).

Some real-life data sets can have missing attribute values. If a tree node requires the value for a particular attribute, and that value is missing, a tree will be unable to give a classification. There are several ways to counter this situation, such as ignoring attributes with missing values, but this wastes information. Breiman et al. (1984) use *surrogate splits* (§2.6.6). Another method uses the probabilities for each outcome as dictated by the examples at the node in question (Quinlan 1986a, 1993b). In other words, the proportion of examples that reach the given node are used to estimate the probability for each branch emanating from this node. To grow a tree in C4.5, the gain is calculated using only attributes with known values, but weighted by the probability that the attribute value is known. Once an attribute is chosen, examples are propagated as follows. For each outcome, assign each a weight, equal to the probability that an example belongs to it. Thus each node may have a fractional weight (if a value is known,

[33] This is indicative of brittle behaviour.

the probability is 1.0). When classifying, one again evaluates all branches resulting in a probabilistic estimate of the likelihood that an example will be classified into each of a subtree's leaves. Utgoff et al. (1990) use an attribute's sample mean at a node, as an estimator of the missing value (also Gams et al. 1987, and Mingers 1989). Utgoff (1995) describes a process whereby examples are pushed down a tree as far as possible, usually to a leaf, but that once a missing value is encountered, the example is simply added to the list at that node, whether there is a subtree or not. The example would then be available for recalculation of node impurity, but not for determining accuracy.

The design of relevant domain attributes is an intensive process and their quality will affect the number of examples required to learn accurate definitions (Matheus 1990). Quinlan (1983) *discovered* attributes automatically by generalising from chess board positions or other configurations known to be 'lost' and some useful attributes were discovered in this way. Gemello et al. (1989) use *data reduction* techniques, where the aim is to reduce the number of examples needed in learning. Some techniques use evaluation functions to select a subset of the potentially usable examples, whilst others 'compress' examples into one (i.e., effectively, generalisation). Evaluation functions may discard potentially useful information, whilst compression adds greater complexity and may over-generalise.

Other systems use weights to identify the more important and/or costly attributes in a domain (e.g. Núñez 1988, 1991), allowing an induction algorithm to have a more informed choice, and thus economise on performance system resources. Tan et al. (1990) reduce cost during learning *and* use. Their algorithm CS-ID3 must be able to ignore features that have not yet been evaluated, and select ones with least cost (as well as greatest information gain). A similar approach is described by Norton (1989). Murthy et al. (1994) select subsets of attributes on which to learn, in order to ignore irrelevant ones. To a certain extent, the use of frames (e.g. Manago et al. 1991) will also achieve

this, by recording only the necessary attributes per object. Indeed, irrelevant attributes can seriously hamper learning efficiency and effectiveness. AQ17-HCI is able to remove such attributes from example descriptions, and possibly replace them with others (Wnek et al. 1994).

### 2.7.5 Pruning

Pruning aims to avoid unreliable subtrees, due to noisy, or insufficient numbers of, examples. Relatively large trees that *overfit* the data occur when an algorithm takes note of chance relationships between examples that do not *generally* occur. These trees do not generalise well and accuracy typically suffers on unseen test cases. On the other hand, *underfitted* trees do not capture all possible relationships and thus may not sufficiently discriminate between classes.

*Pre-pruning* stops tree growth during training whilst *post-pruning* tries to improve accuracy after a tree is grown. Schaffer (1991) identifies occasions when overfitting might not decrease accuracy and thus pruning would be erroneous. This topic does not directly impinge on our thesis and so it is sufficient to say that Schaffer implies that generally one should be careful when applying the 'simple-is-best' heuristic. That is, where noise is absent and only *relevant* attributes are used, pruning may lead to suboptimal trees.

Quinlan (1987b) covers several methods for pruning or simplification of decision trees:-

- cost-complexity pruning
- reduced-error pruning
- pessimistic pruning
- production-rule reformulation

A tree's *cost-complexity* is defined as the sum (see Quinlan 1987b, Breiman et al. 1984,

Niblett 1987) :-

$$\frac{M}{E} + \alpha \times L(T)$$

where $M$ is the number of misclassified examples (out of $E$), $L(T)$ is the number of leaves in a tree $T$, and $\alpha$ is a parameter. Initially a tree is grown from the training data. A set of trees is then generated from this by successively replacing subtrees with leaves, from the bottom up. The pruned tree is generated as follows. Pruning a subtree $T'$ may result in $M'$ further errors, but the tree is smaller by $L(T') - 1$ nodes. A new tree has the same cost-complexity if $\alpha = \frac{M'}{E * (L(T')-1)}$, and so one examines all subtrees and picks the one which minimises $\alpha$. The subtree concerned is then pruned, i.e. replaced by a leaf. Finally one tree is selected for use by classifying a test set (size $E'$) and picking the smallest subtree whose accuracy is within a standard error of the least number of errors in all subtrees[34] . Breiman et al. (1984) use a similar pruning technique, but use $K$-fold cross-validation, obviating the need for a separate test set.

*Reduced-error pruning* (Quinlan 1987b) is similar in that it utilises a sequence of successively smaller, pruned trees. Each subtree is pruned if and only if it results in the same number of, or fewer, errors. The process stops when the tree can no longer be pruned without increasing the number of misclassifications. Again this requires a separate test set and can remove special case leaves that represent statistical outliers.

Pessimistic pruning (Quinlan 1987b) uses resubstitution, and as such does not require a separate test set. For a given leaf of $E$ examples, if $M$ of them are misclassified, then the resubstitution estimate of the error rate is $\frac{M}{E}$ which can be overly optimistic. If $T'$ is a subtree of tree $T$, with $L(T')$ leaves and $\Sigma M$ and $\Sigma E$ are the sums over $T'$, as above, then in this scenario it is more likely $\Sigma M + L(T')/2$ of the $\Sigma E$ examples will be misclassified. If $T'$ is replaced by a leaf and misclassifies a further $M'$ examples, then $T'$

[34] Standard error: $se(M') = \sqrt{\frac{M' \times (E' - M')}{E'}}$

is pruned away if $M' + \frac{1}{2}$ is less than one standard error from $\Sigma M + L(T')/2$.

Individual tree paths can be considered to be individual rules (e.g. Quinlan 1993b). This simplification method consists of transforming the tree to rules, removing 'unnecessary' terms from within rules, followed by the removal of 'unnecessary' rules. Each rule is simplified by removing conjuncts (i.e. former tree test nodes) that are judged to be irrelevant by way of a statistical test (Fisher's exact test - see Quinlan 1987b). In C4.5 (Quinlan 1993b) MDLP (§2.7.1) is used to minimise rule sets. Finally, a certainty factor is added based on the number of examples satisfying the left-hand side of a rule and are correctly classified.

The *chi-square* test is often used to aid pre-pruning (e.g. Quinlan 1986a, b; Mingers 1989 also uses it for attribute selection), but becomes unreliable when there are few examples at a leaf, something that can occur frequently unless one has large sets or simple concept definitions. This may be avoided with the use of "Fisher's exact test" (Niblett 1987). Pre-pruning might also preclude splits which do not directly improve node purity (e.g. parity concept) but where a *combination* of tests will. Post-pruning avoids this but can add considerably to run-time. The chi-square test indicates whether an attribute is statistically dependent on a class and it is therefore useful for classification, i.e. one can tell whether one variable is dependent on another. Another approach is described by Van de Velde (1989, 1990) where topologically minimal trees are sought (see §3.1.3). Gams et al. (1987) mention others, for example, stop splitting when node 'weight' (in terms of the numbers of examples) falls below a threshold.

### 2.7.6 Constructive Induction

Constructive induction introduces new terms into the output language. As Muggleton states (1987), the idea is to introduce *meaningful* terms which will then aid the classification process. In Utgoff's terminology (1986), constructive induction constitutes *dynamic*

bias (see also Rendell et al. 1987). Ideally, a constructive induction algorithm can overcome inadequacies stipulated by inappropriate bias (i.e. representation language). New terms can lead to more concise concept descriptions but can also lead to a large, complex search space. Examples include (Michalski 1980, 1983), DUCE (Muggleton 1987), AQ15 (Michalski et al. 1986a), CIGOL (Muggleton 1988, Muggleton et al. 1988), FRINGE and GREEDY3 (Pagallo et al. 1989, Pagallo 1989), CITRE (Matheus 1990) variations on AQ17 (Wnek et al. 1994, Thrun et al. 1991), (Schlimmer 1987b), and (Donoho et al. 1995).

Issues such as *when* to use constructive induction and how to *guide it* are addressed by Watanabe et al. (1987). Searching for new attributes can be a computationally expensive process[35] . The algorithm LAIR (Watanabe et al. 1987), relies on the knowledge-base to constrain construction of new predicates, e.g. if the current hypothesis contains a predicate $P$, and the knowledge-base contains a rule $P \rightarrow Q$ then replace $P$ with $Q$. The knowledge-base is partly provided (a priori), partly learned and is used as above to deductively constrain induction.

Pagallo et al. (1989) and Matheus (1990) note that trees can suffer from the *replication* problem where attribute test sequences are repeated across subtrees. This can lead to excessive fragmentation of the example set and thus the need for more examples to ensure reliable induction. FRINGE (Pagallo 1989) builds a tree using the primitive attributes, then examines the test sequences near the positive leaves (fringes) and constructs Boolean combinations of these tests to form new attributes. Examples are then redescribed in the new terms, whence the tree is rebuilt and the process reiterates. This iteration halts when no more attributes can be added or a maximum is reached. In CITRE (Matheus 1990) constructed features are evaluated and 'useless' ones are discarded to constrain search times and so forth. Evaluation uses performance criteria such as accuracy and

[35] few attributes are needed before a search space becomes untenable

frequency of use in hypotheses etc.. STAGGER (Schlimmer et al. 1986b) adds new attributes to a language which correspond to the concept descriptions attained part way through its incremental learning. Donoho et al. (1995) use constructive induction to repair a deficient *theory* (§2.5.8).

### 2.7.7 Incremental Induction

Machine learning has been recognised as a method for alleviating the knowledge engineering bottleneck. It seems prudent therefore, that knowledge-bases be *incrementally* updated, not regenerated from scratch as new information becomes available (Schoenauer et al. 1990). Incremental concept learning is the process by which concepts are induced over time, with a possibly incomplete training set which is periodically enlarged as more data becomes available. Langley et al. (1987) are concerned with 'plausible' models of learning for humans and propose three dimensions to theories of learning, including *incrementality*. Humans must learn incrementally because of the sequential nature of information, i.e. one rarely has a complete set of facts before reasoning is required (Reinke et al. 1988). Michalski (1986) states :-

> "The meaning of human concepts is dynamic; it changes with time and adapts to new contexts and requirements. Novel concepts are continuously being created and developed, and some are being outgrown."

One is also limited by a finite memory, requiring information to be processed as and when it is acquired. Thus learning is *non-monotonic* (Bain et al. 1987, Schoenauer et al. 1990), as descriptions are *revisable*.

> "... knowledge revision is typically much less expensive than knowledge creation." (Utgoff 1995)

Particular benefit is to be found with large data sets, especially those with many complex and noisy features. An inefficient algorithm can make induction in these domains all but intractable (e.g. see below, and Shen 1992). Algorithm efficiency is also tackled in (Sutton et al. 1993) and (Musick et al. 1993), amongst others.

Examples include AQ15 (Michalski et al. 1986a, Kodratoff et al. 1990), AQ11 (Michalski et al. 1980), GEM (Reinke et al. 1988), LEX (Mitchell et al. 1983), CIGOL (Muggleton 1988, Muggleton et al. 1988), ID4 (Schlimmer et al. 1986c), VBMS (Rendell et al. 1987), COBWEB (Fisher 1987a, b), CLASSIT (Gennari et al. 1989), ID5(R) (Utgoff 1988a, 1989a, 1989b), IDL (Van de Velde 1989, 1990), STAGGER (Schlimmer et al. 1986a, 1986b), UNIMEM (Lebowitz 1987, 1888), PT2 (Utgoff et al. 1990), IB$x$ (Aha et al. 1991, Aha 1992), ITI (Utgoff 1995) and others (Vrain et al. 1988, Sebag et al. 1991a, b, Holte 1989, Schoenauer et al. 1990, Cai et al. 1991, Kubat et al. 1991, Conroy et al. 1994, 1995).

Advantages offered by an incremental approach include its ability to allow new information to be added to a concept representation without having to restart the induction process. Rebuilding a whole tree when, in all likelihood, a small change is required, is an obviously inferior approach given the often copious, complex and volatile nature of real life data. Incrementality can thus allow an induction algorithm to circumscribe the combinatorial explosion that can be inherent in large, multi-attribute example sets (e.g. Musick et al. 1993). That is, non-incremental algorithms are restricted to 'one-shot' learning and may attempt to examine the whole example set at once. With incrementality, examples are handled a few at a time instead of in toto (Manago et al. 1991, Schoenauer et al. 1990, Sebag et al. 1991a, Lebowitz 1988, Shifu et al. 1992), thus obviating the need for a 'window' on the example set (Quinlan 1979). As training sets grow, being restricted to a subset, or window, for computational reasons is not the ideal answer and has been shown to increase run-times and is of no real benefit (Wirth et al.

1988)[36] . Blythe (1988) suggests the combination of both non-incremental and incremental learning, that is, the initial learning from a large collection of examples followed by the incremental adjustment of the concept definition.

Less concern about the number and size of examples (e.g. Sebag et al. 1991a) can relax the need for background knowledge required to constrain the search for an effective representation, thus making the algorithm more generally applicable. Tadepalli (1989) describes *Lazy-EBL*, where an incomplete explanation is formed and then refined incrementally. This results in a reduced learning cost, first of all from the reduced deduction and secondly because partial plans are only corrected if they prove to be inadequate for the performance task.

Incremental learning with perfect memory (Michalski et al. 1986a), means that all examples seen so far are retained. Partial memory systems will retain say, only prototypical examples, exceptions, or even merely concept descriptions. One can usually discard previously seen data if back-tracking is allowed, as there is no need to retain it for a possible 'rerun', a distinct advantage if machine space is limited.

Combined with some appropriate form of pruning, incremental learning can track *concept drift* (Schlimmer et al. 1986a, 1986b), or *non-stationary concepts* (Sutton et al. 1993, Kubat et al. 1991). Utgoff (1989a) suggests that new examples should replace older ones where appropriate (i.e. if a new example has a different class).

Incremental learning can help to induce smaller trees (Utgoff 1989b) and focus a learning algorithm on misclassified examples, or exceptions e.g. ADECLU (Decaestecker 1991), $\widehat{\text{ID5}}$ (Utgoff 1988a). Indeed, ID3 can result in "empty" leaves (if there is more than two classes - Quinlan 1990c), as all attribute values are assumed to exist in all branches of a tree. This does not occur in incremental settings as branches are added as

[36] In fact, with a few restrictions (e.g. only iterate if accuracy increases) and alterations (e.g. pick uniform class distributions), Quinlan (1993b) notes that accuracy may be improved, but time also increases considerably.

needed.

Incrementality can also allow bi-directional searching which is important when poor choices may affect a concept and allows an algorithm to recover from them, e.g. ID4, ID5 (§3.1.2). As Schoenauer et al. (1990) note, incremental algorithms can allow the improved handling of noise (decisions are revisable) but convergence upon a solution may not be possible due to the oscillatory application of operators (e.g. ID4 and Decaestecker 1991). Bi-directional searching allows quasi-backtracking without its inherent overheads (Langley et al. 1987). The ability to backtrack can thus render an algorithm less susceptible to the order of presentation of examples and to the effects of noise (Lebowitz 1988, Gennari et al. 1989, Decaestecker 1991, Schoenauer et al. 1990, Sebag et al. 1991b).

Fisher (1987b) and Schlimmer et al. (1986c) propose that incremental systems should be evaluated over three criteria:-

- the cost of incorporating examples into the classifier
- the quality of the classifier
- the number of examples required to converge on a stable classification

In the next chapter we elucidate further the incremental induction of decision trees and offer an appraisal of certain algorithms. In Chapter 4, we explain our approaches for dealing with the first item above, i.e. concept update cost and efficiency, with regard to, primarily, the second item, concept quality.

# Chapter 3

# Incremental Decision Tree Induction

## 3.1 Incremental TDIDT

ID3 (Quinlan 1979) is a well known tree induction algorithm which has been extended in a number of ways (e.g. Quinlan 1993b). Two of these extensions, viz., ID4 (Schlimmer et al. 1986c) and ID5 (Utgoff 1988a), are said to be incremental in that examples can be added piecemeal to an existing tree. In this section we explore in more detail these, and other algorithms.

### 3.1.1 ID4

ID3 can be made to process incrementally simply by restarting the whole tree generation process from scratch whenever new examples arrive. Obviously this is a worst-case scenario and significant improvement in tree update time is desirable. This worst-case algorithm is termed "INCREMENTAL-ID3". In fact, Gennari et al. (1989) deny that such extensive reprocessing of training data can be labelled as incremental. A truly incre-

mental algorithm, ID4, is as follows. Consider a tree built by ID3 where each test node has a number of potential test attributes, i.e. the algorithm is faced with a choice for the test attribute. In ID4, for each potential test attribute at a node, a count is maintained for each attribute value, in terms of the example classes. In other words, if one assumes two classes (positive and negative) then there are two entries for each attribute value per decision node. Each entry is a count of the number of examples seen so far that have the particular attribute value in mind. Mingers (1989) terms these *contingency tables.* Subtrees are grown as in ID3 but with an added statistical *significance* test ($\chi^2$). This is a form of pre-pruning (§2.7.5) which will only allow a 'split' if the implied relationship is not thought to have arisen by chance (see also Quinlan 1986a). Each time a new example is added to the tree, it starts at the root and follows a path according to its value for the current test attribute. At each node, the entropy calculations which first picked the current, or *installed* (Utgoff 1995) test attribute are recalculated. Thus one can immediately determine whether the current test attribute is still the 'best' choice. If it is, the tree is left unchanged and the example propagation continues to the next level in the tree path. Otherwise the current test attribute is deemed to be suboptimal (with respect to entropy), i.e. another attribute would give greater information gain at that point in the tree. The subtree is discarded and replaced by the new test attribute. The result is to replace several tests with one. To rebuild a discarded subtree to its previous detail, further training instances which will be propagated to this new leaf are required.

In a 'well behaved' environment (cf. the parity problem), the tree stabilises from the top-down, level by level until convergence upon an ideally (quasi-) optimal tree occurs. Schlimmer et al. (1986c) analyse the efficacy of their algorithm in a number of ways. In a tree with root $T_0$, level $T_i$ must stabilise before $T_{i+1}$ can and this leads to a worst case estimate of using all $E$ training examples, *per level*, to construct a given tree (ID3 would require $E$ in total).

Schlimmer et al. also make use of *smarter* versions of ID4 and incremental ID3, namely $\widehat{\text{ID4}}$ and $\widehat{\text{ID3}}$. These change a tree only when a *misclassification* occurs, i.e. a training example arrives at a leaf of a different class. This utilises 'near-miss' examples assumed to be close to the (geometric) concept boundary in concept-space[1] . In the experiments which Schlimmer et al. perform, both ID4 and $\widehat{\text{ID4}}$ required substantially more examples to converge to a 'complete' tree; discarding suboptimal subtrees loses information which could otherwise be put to good use. The accuracy of $\widehat{\text{ID4}}$ also lagged somewhat behind the other algorithms (i.e. took more examples to achieve the same score). This implies that training only on misclassifications is not the ideal solution with regard to accuracy and 'effort' and that some correctly classified examples are useful too. In terms of updating concept descriptions, both versions of ID4 fared considerably better than an incremental ID3 (i.e. much less computation).

Utgoff notes (1989a) that ID4 is unable to learn some concepts (in the context of unlimited examples and a desired accuracy). This occurs when ID4 cannot decide in which order test attributes should be arranged and repeatedly discards subtrees, thus preventing the tree from stabilising (Utgoff adds that $\widehat{\text{ID4}}$ was worse still, in the test performed).

### 3.1.2 ID5

Usually, not all of a discarded subtree needs to be changed, and so ID4 is at a disadvantage. In other words, if the tree were rebuilt from scratch then one could find that most of the tests in the purportedly suboptimal subtree may well remain as they were. ID5 (Utgoff 1988a) rectifies this by not discarding subtrees but *restructuring* them. The idea is to identify the best test attribute as before and then attempt to *pull* this attribute up from a lower position in the tree to become the new subtree root. Again, ID5 achieves

[1] and thus more informative for specifying the said concept boundary.

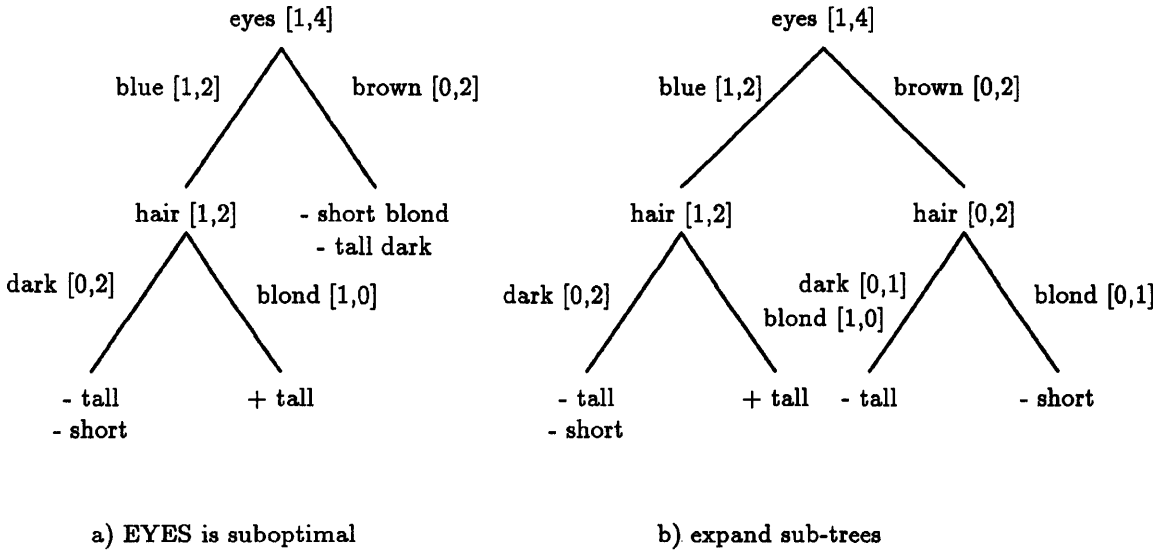



Figure 3.1: ID5 tree manipulation - initial step plus expansion

incrementality by maintaining statistics at each decision tree node that summarise class distributions, in terms of attribute values. As with ID4, ID5 can then judge whether a subtree has become suboptimal with respect to entropy and then restructure in order to keep the most informative attributes nearer the root of the tree.

ID5 is a *perfect memory* algorithm in that all examples are retained. This is achieved largely implicitly through the use of the tree structure. Therefore, at a given leaf, only the attributes not tested so far are required to be stored, the rest are stipulated by the path back to the root.

As an example is added to a node, the usual counts are incremented (termed *instance count additions* or ICAs by Utgoff) and the potential test attributes are rechecked to see which is best. If no revision is needed, the algorithm progresses to the next level, until a leaf is reached. If there is only one class, the portion of the example not implicit in the tree is stored. If the leaf is impure, the tree is grown as in ID3. If at some stage a revision is needed, a subtree is first expanded to include the new 'ideal' test attribute in all paths. Examples are expanded to include the implicit parts. The new best attribute

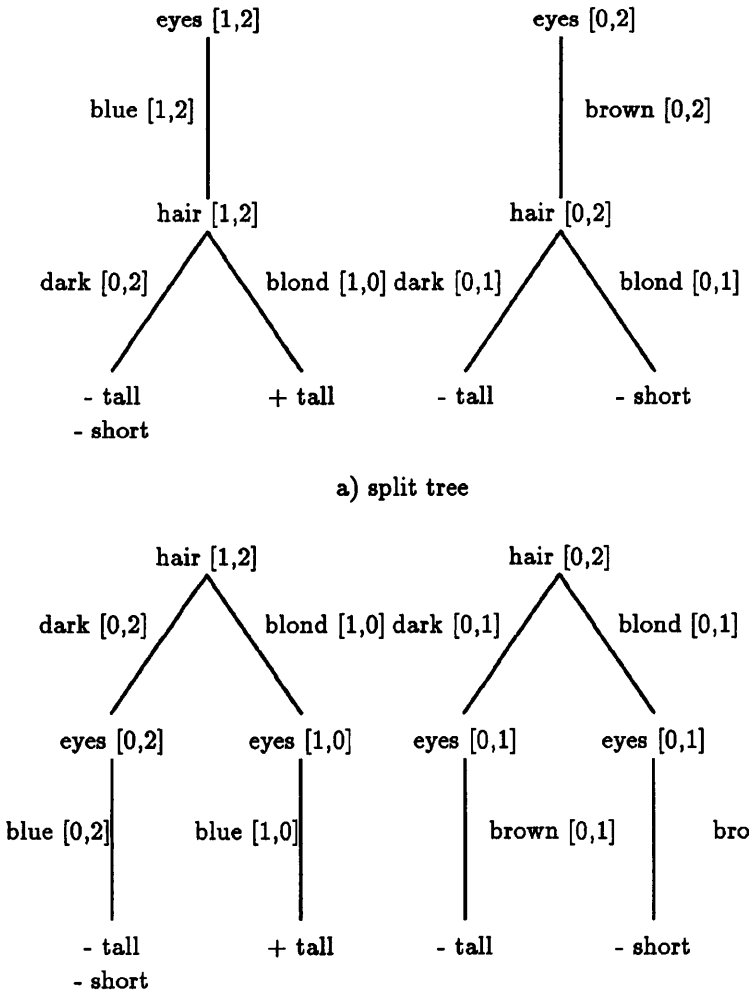



Figure 3.2: ID5 tree manipulation - split and swap

is then recursively pulled to the root of each subtree of the old test attribute, whereupon it is swapped with its parent, that is, the suboptimal test attribute is pushed down one level. Duplicated subtree roots are then merged and subtrees of one class are contracted to a leaf. This process is depicted in the following figures, adapted from (Utgoff 1988a). Class counts are given in '[ ]' after an attribute, and leaf classes are indicated by '-' or '+'. Figure 3.1(a) depicts the initial stages, where "eyes" has become suboptimal and is to be replaced with "hair". Figure 3.1(b) shows the expansion of the leaf below "brown".

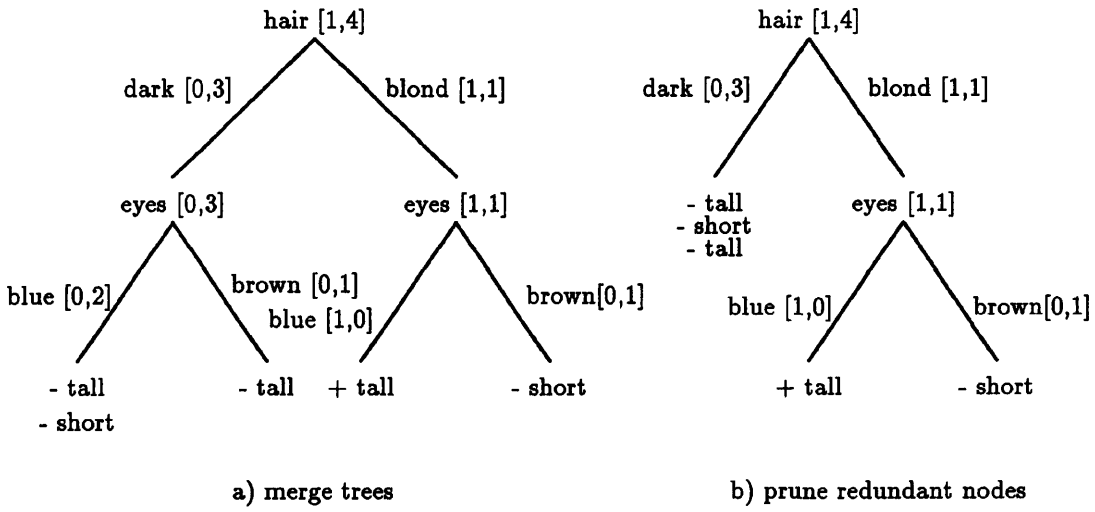



Figure 3.3: ID5 tree manipulation - merge and prune

The next step recursively pulls "hair" to the root of each subtree, which is not necessary here as it is already so. Figure 3.2(a) shows the tree split into separate subtrees, one per child of "eyes". Figure 3.2(b) shows the attribute swap, whilst figure 3.3 depicts the final merge (a) and prune (b). Thus ID5 has the operators 'split' (tree growth), 'prune' (remove redundant nodes from the merging process), and 'transpose' (restructure), (Van de Velde 1990).

In the first version (Utgoff 1988a), the process would now stop, possibly leaving sub-optimal subtrees below the 'swapped' test. That is, the subtree paths are not rechecked to see if there are more restructures which should be performed. In a subsequent version, i.e. ID5R (Utgoff 1989a), the subtree below the new test attribute is recursively rechecked to ensure the most favourable sequence of tests is in place. This ensures ID3-equivalent trees and thus the most informative attributes occupy the highest positions in the subtree.

Utgoff performs a worst-case analysis of the effort required to build a tree and notes that the frequency of tree revision will have a significant effect on efficiency estimates (see also below). Again, versions which update only on misclassifications in order to

save on computation were also used, i.e. $\widehat{\text{ID5}}$ and $\widehat{\text{ID5R}}$. It is noted (Utgoff 1989a) that all incremental algorithms built trees smaller than ID3 (for the multiplexor domain), although more examples were used. This begs the question, why does Utgoff present ID5R as an incremental ID3-tree-equivalent algorithm when ID3 is obviously suboptimal? These and other points will be discussed below, and in the next chapter.

### 3.1.3 IDL

ID3 is biased towards the induction of small trees, and the design of IDL (Van de Velde 1989, 1990) reiterates this. IDL is based on ID5(R) and is thus incremental. An additional heuristic enables IDL to find trees which are often optimal in size[2] , in many domains, requiring less computation and fewer training examples. This search for a *topologically minimal* tree stems from similar motivations as tree *pruning*, i.e. redundant tests. IDL searches from the leaves up, using a heuristic based on a tree's structure, not on a statistical method, and aims to avoid the obsolete repetition of tests within trees (see also Pagallo 1989).

Van de Velde claims (1990) that ID4 and ID5(R) often find suboptimal trees (with respect to size)[3] . An example is *covered* by a tree path if all tests on the path are satisfied by the example, including the class at the leaf. Such a path is unique for the example, but there are other *partial* paths which may cover part of the example. These are found in a bottom-up fashion by starting at leaves of the *same class* as the example in question. One then climbs successively upwards if the parent decision node test is satisfied by the example. The *topological relevance* ($TR$) for an attribute $A$ is then defined as the number of times $A$ appears in the partial and full paths covered by the example in question. $TR$ gives an indication of the importance of $A$ for classification.

[2] i.e. contain fewest nodes

[3] although it has been noted that so too can IDL (Elomaa et al. 1990)

Thus attributes with high $TR$ values would be preferable in higher positions in a tree and may render other tests obsolete. That is, highly discriminatory tests often result in smaller trees. The tree manipulation procedures used in ID5(R) (§3.1.2) are now used to push preferred attributes further up the tree and consequently minimise $TR$ scores. IDL starts in the same way as ID5 etc. with an empty tree which is grown incrementally using measures such as entropy (§2.7.1). IDL uses training examples to guide its search for less efficient tests, which are then pushed down the tree and possibly pruned away. Using the current example keeps this search relevant. Van de Velde conjectures that in most cases IDL will find a topologically minimal tree and will then adhere to it, unlike ID4/5. If no such tree exists, IDL will not converge to a unique tree, and thus *limit-cycles* through equivalent, suboptimal trees (i.e. of the same size). As noted above, IDL sometimes converges to non-topologically-minimal trees (Elomaa et al. 1990), although this may be dependent upon example ordering. Elomaa et al. conjecture that the failure to converge to a topologically minimal tree may lead to a failure to converge to *any* tree if input is repeated indefinitely.

IDL was also found to reduce run-time, entropy calculations, tree growths, prunings and especially tree restructures over ID5R. Van de Velde notes that IDL may well be sensitive to noise due to its reliance on single examples.

### 3.1.4 ITI

Utgoff (1995) describes two algorithms for efficient restructuring of decision trees, one of which (ITI) is incremental, and is descended from ID5R. The other (DMTI), aims to use measures of tree quality more explicitly. Utgoff utilises binary trees, which also list attributes, values and their classes, in binary *search* trees at each decision node, to aid efficient update. ITI implements multi-phase example addition, as do QUASI_JITTER and JITTER (Chapter 4). Utgoff notes that such a policy allows a number of "training

modes". A "lazy" mode allows one to add several examples and only update counts (etc.) once, at the end. This in itself could save much processing, i.e. one only rechecks the tree for revisions when an up-to-date tree is required for some purpose. Use is made of the Minimum Description Length Principle (§2.7.1) to prune subtrees which are insufficiently compact and/or accurate. However, any pruned subtrees are not discarded, but simply marked as pruned. This "virtual pruning" preserves information for further updates, but provides a (hopefully) less over-fitted subtree for use in classification.

Use is also made of a "stale marker" at each decision node (similar to ID5_DELAY's and PSEUDO_JITTER's counters, but see Chapter 4). When an example is added to a path, each node is marked as stale. This marker is only removed if the recalculation of the impurity function indicates that a tree restructure is required. Utgoff also describes an efficient method for processing for continuous attributes. Each value is listed in order at a node, tagged by the class of the example from whence it came. Then each pair of adjacent values of a different class gives rise to a possible cut-point, midway between them.

DMTI is a non-incremental tree induction algorithm again described in (Utgoff 1995). This algorithm builds a tree using a typical impurity function (gain ratio) and then evaluates the whole tree, possibly revising it, based on a global "direct metric", or quality measure. This allows one to alter the bias required for a tree, i.e. one simply alters the direct metric. To a certain extent then, Utgoff splits some of our aims across two algorithms. We show how these may be combined in one.

### 3.1.5 Others

Cockett et al. (1989) use a similar approach to ID5 and IDL to reduce trees in algebraic form to a minimum size. The algorithm attempts to push non-essential attributes down the tree and removes them from the leaves using *syntactic* relations (e.g. $q(x, x) = x$,

says that a node $q$ with identical children can be reduced to a single leaf). Cockett et al. note however, that such an *irreducible* tree is not guaranteed to be *simple* as some redundancy can still remain.

Crawford (1989) introduces some extensions to the CART algorithm (Breiman et al. 1984), enabling incremental processing of *binary* trees. A new example is added to the tree and the new impurity is calculated at each node. The optimum split may now have changed, which may cause the example to be sent down the *alternate* branch, i.e. not the one it would be sent down with the current split. So, CART computes the decrease in impurity for both cases, the current split $s$ where the example behaves as it should, and a hypothetical split $s'$, where the example takes the alternate branch. If the impurity decrease for $s$ (with the extra example) is less than that for $s'$ (plus example), CART searches for a new test attribute. If the new attribute is not $s$, the subtree is pruned and replaced with the new attribute, whereupon CART *rebuilds* both child subtrees.

Crawford's experiments revealed a large number of tree restructures (many unnecessary), at least partially caused by attributes being dependent upon one another and by noise, with new trees often being virtually the same as the old. In light of this, Crawford amends the procedure with extra statistical calculations to determine whether a new suggested split is actually worthwhile, with improved performance.

Bhandaru et al. (1991) use a general tree-like structure called an *HC-Expression*, where successive levels are positive and negative exceptions. To classify an example, one starts at the root and finds a lower node which 'covers' the example, then one checks to see if the example is included in the exceptions one level below. The example's class is then the class of the last covering node. That is, each node is described by a 'cover', as in the AQ algorithm (§2.6.2). The *quality* of nodes is judged by an accuracy-like measure ($\frac{e_i - e_{i+1}}{e_i}$, where $e_i$ is the number of examples satisfying node $i$, and $e_{i+1}$ is the number satisfying the node below). If node quality is poor, then a node is split (into

value subsets). The algorithm also updates on exceptions only, and the authors hold that the structure is easy to change, and so is useful for incremental learning.

## 3.2 An Appraisal of Incremental TDIDT

Below we describe how present algorithms can wander in a relatively uninformed fashion through the concept space, wasting search resources in the mean time. We then characterise the major computational elements of induction for ID4 and ID5.

### 3.2.1 An Evaluation of the Efficiency of ID4 and ID5

ID4 and ID5 can often change trees more frequently than necessary to maintain tree quality (e.g. accuracy). This can have a knock-on effect causing, for example, more entropy calculations, resulting in greater run-times. Furthermore, the computational cost of an *individual* restructure may be prohibitive, as it is recursive and may involve all levels of a tree. Therefore, as Utgoff (1988a) notes, the expected frequency and computational cost of tree revision, and the implied impact on further training, are extra efficiency considerations for ID4 and ID5, over and above those for ID3. To our mind, entropy calculations are also intensive, time consuming operations which should be avoided if they are not absolutely necessary. Indeed Utgoff (1989a) notes that a significant expense for ID5(R) is the number of such calculations. Manago et al. (1991) note ID3 does many "useless" entropy calculations which "cannot be afforded" when data becomes complex.

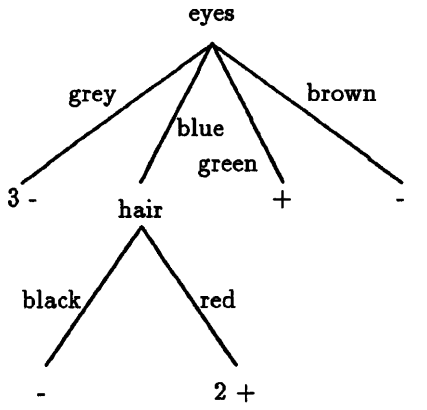
Best splits are always tentative (Van de Velde 1989), especially during the early stages of a tree's development, as there is less information on which to base decisions. Entropy can cause a tree node to be swapped for another (or discarded in ID4) when in fact this change does not increase the tree's accuracy nor decrease its complexity but simply gives greater information gain. Cockett et al. (1989) describe entropy as a

*local* selection method which does not necessarily optimise *global* tree quality, i.e. it is a 'greedy' algorithm (Murthy et al. 1995). For the majority of cases attribute selection by entropy may well be beneficial but there are often *functionally* equivalent trees that entropy will ignore because it does not take explicit account of global tree quality, viz. accuracy, complexity and so on. We define functional equivalence primarily to mean trees of the same accuracy and size.

Utgoff (1989b) also shows how the size of a tree produced by ID5 can differ wildly. The example given concerns the addition of three examples, and tree size fluctuates from 87 to 111 to 87 nodes (we would be interested to see how the accuracy differs here as well). Accuracy and complexity may well improve upon a restructure, but it is evident this is not always the case. It appears that *intermediate* accuracy[4] does increase for an ID5 tree, although not monotonically (but see below).

*Aborted restructures* occur when ID5 recalculates entropy for every potential test attribute at a node, only to find the current one is still the most favourable with respect to information gain. As with all restructures, such information is wasted if a change (attempted or otherwise) does not improve tree accuracy, size, and so on (not to mention the manipulation involved in an actual change). $\widehat{\text{ID5}}$ reduces the number of actual and aborted restructures by updating a tree only when examples are misclassified (Angluin et al. 1983, term this *conservatism*). However, tree manipulation and the act of entropy calculation is still relatively blind as it is guided solely by information gain. Suppose one has the following decision tree, as generated by $\widehat{\text{ID5}}$ (figure 3.4(a)). This is a tree generated part way through an artificial example set, examples being added one at a time to the tree. The example file is an arbitrary extension of one of Quinlan's (1983) training sets with 10% attribute noise (for the full set see Appendix C). The tree is trained on a randomly selected two-thirds of the example set and tested with the remainder. The

[4] Accuracy as measured (by a test set) after the addition of *each* training example.




a) examples = 8

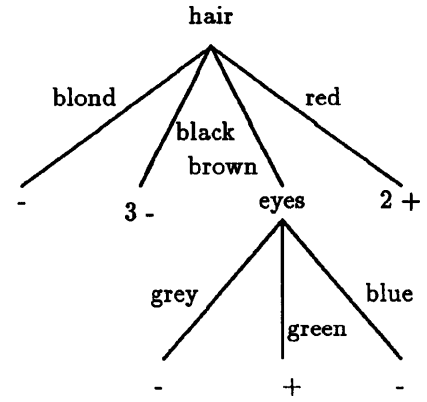
restructures = 0

nodes/leaves = 7

entropy calcs = 11

aborted restructures = 2

accuracy = 60%




b) examples = 9

restructures = 1

nodes/leaves = 8

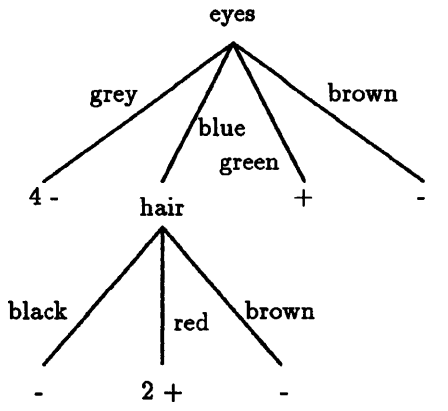
entropy calcs = 14

aborted restructures = 2

accuracy = 40%

example no. 9 =

eyes=blue,hair=brown,height=tall,-




c) examples = 10

restructures = 2

nodes/leaves = 8

entropy calcs = 22

aborted restructures = 2

accuracy = 60%

example no. 10 =

eyes=grey,hair=red,height=tall,-

Figure 3.4: $\widehat{\text{ID5}}$ trees at three stages

accuracy is measured at 60.0%. In figure 3.4(b), after the addition of one more example, $\widehat{\text{ID5}}$ has decided to change the structure of the tree and put the *hair* attribute at the root. The accuracy has dropped to 40.0% and the number of nodes/leaves has actually *increased* from 7 to 8. The change certainly appears to be completely unnecessary. Our earlier point about entropy not directly shadowing tree accuracy or complexity is brought home even more forcibly when one considers that on the addition of one further example (fig. 3.4(c)), $\widehat{\text{ID5}}$ again restructures, putting the *eyes* attribute *back* at the root.

The ID5R algorithm (Utgoff 1989a) ensures ID3-equivalent trees are produced. First of all we take issue with such an objective, given the previous comments on entropy (end of section 3.1.2). Van de Velde (1989, 1990) uses IDL (§3.1.3) to induce (quasi-) topologically minimal trees, in a global sense, lending more credence to our point that entropy on its own is not ideal. Secondly, given that such a protocol *is* beneficial, any unnecessary restructures will further compound loss of efficiency due to the *recursive* restructuring. In our example above, swapping the root attribute would still occur, and potentially *all* others below it, only for it to be replaced on the next example. Thus one must be even more careful with ID5R's restructures. ID4's restructure algorithm will increase the incurred computational costs as it discards whole subtrees, relying on further examples to rebuild them. Thus, unless one has unlimited or very large numbers of examples, such a rebuild (as above) could easily preclude the second change back to the original tree. Similarly, a brute-force incremental ID3 does not even have this choice and would rebuild the whole tree from scratch whether a restructure is required or not.

In their paper on "on-line learning", Sutton et al. (1993) criticise ID4/5 for unbounded memory and computation requirements, which can cause problems for learning algorithms which are tightly coupled to the performance system.

The induction algorithms presented thus far can be described as relatively simplistic, that is, the basic algorithms utilise only categorical attributes with 'small' numbers of

values, no missing values, and so on. On studying more complex versions of tree induction algorithms (e.g. later versions of ID3, CART, AID etc.), one can see that in order to cope with say, continuous values, further complex calculations are required. Thus for a continuous attribute, one must also find the best partition of its range and analogous situations can be identified for the other cases. In addition, Utgoff (1995) notes that the cost of updating a tree incrementally, with an example of categorical attributes, is independent of the number of examples seen so far. On the other hand, for numeric attributes, cost is *not* independent of the examples seen. This is due to the recalculation of cut-points which depend on the number of examples at a node. With such problems one must be doubly careful where and when to allow tree changes and/or entropy calculations as such calculations can add appreciably to computation. Noise and divisive problem classes (e.g. parity, multiplexor, etc.) may exacerbate these facets. For example, given an unlimited supply of examples, ID4 oscillates and does not converge to a stable tree for certain example sets[5] (Utgoff 1988a).

### 3.2.2 Conclusion

The following points are therefore noted:-

- splits are often tentative
- entropy is local and therefore does not take direct note of global quality, e.g. accuracy
- entropy is subject to random perturbations in attribute-value class distributions
- restructure costs can be prohibitive
- restructures have implications for further training

[5] due to (quasi-) backtracking

- restructures can cause size and accuracy to fluctuate wildly
- restructures can cause the non-convergence to a stable concept
- aborted restructures can cause further efficiency losses
- tree accuracy does not increase monotonically during training

Noise, multiple and/or missing values, and irrelevant and/or continuous attributes may compound the above problems. In addition, we note that:-

- ID3 equivalence is not necessarily beneficial per se
- ID5R is recursive, therefore further constraints on restructure frequency may be required
- $\widehat{\text{ID5}}$ accuracy can lag behind ID3/ID5R, therefore training on misclassifications only is not ideal
- IDL adds to inherent computation as it also prunes
- ID4 discards useful information

Thus we conclude that ID4 and ID5's search of the concept space is relatively uninformed and potentially wasteful. Certain versions ameliorate these effects, such as $\widehat{\text{ID5}}$ and IDL, but do not go far enough. Real world problems often include noise, continuous and missing values, etc., and as such could prove potentially intractable to current algorithms, within, or regardless of, a given run-time.

### 3.2.3 Worst-Case Analysis of ID4 and ID5

There are a number of aspects of incremental tree induction which can be reduced with a view to increasing the efficiency with which an algorithm works. These include :-

- the number of entropy calculations[6]
- the number of tree restructures
- the number of aborted restructures
- the computational cost of individual restructures (including extent)
- the impact of restructures on the need for further training
- tree size/complexity[7]

Here we show how some of these factors influence the computational process of tree induction (terminology is defined in Appendix A.2). Below, we give estimates of the computational effort required for various aspects of tree construction, in terms of *basic computational steps*. These consist of mathematical operations such as addition, subtraction etc., comparisons of characters, memory allocation and so on, and are all assumed to be equivalent in terms of the resources required to effect them.

### Entropy

First we describe entropy calculations in terms of basic mathematical operations. Each $\sum_{i=1}^{C} p_i \log p_i$ can be described as a sequence of $C$ additions, $C$ multiplications, $C$ subtractions, $C$ logs and $C$ divisions, which gives $O(C)$ basic operations. Each full entropy calculation must repeat this for each of $A - d$ attributes at the current node, each of which has up to $V$ values, together with a constant number of other additions, subtractions, multiplications and comparisons. This gives a total of $O(A.V.C)$ every time one calculates entropy at a node.

Entropy may in fact be calculated *incrementally* by table lookup, which saves on computation. If one saves the $p_i \log p_i$ total for each class at each node, then it is possible

[6] or whatever attribute selection procedure is used

[7] Tree size is important because it is more time consuming to update and manipulate larger trees.

to ascertain the new value for each class that will change upon the addition of an example. Hence the information available at the node is more easily calculable. This can be effected through the use of tables of such incremental values. Entropy recalculation is then reduced to $C$ lookups and $C$ subtractions, i.e. $O(C)$. One must still perform these calculations for each attribute value which gives $O(A.V.C)$. Obviously this algorithm adds to storage requirements, i.e. one real per class per node, plus whatever tables are required. It would also seem that there are occasions when entropy will have to be recalculated from scratch anyway, such as the addition of new branches to an existing node, the addition of an example of a new class, the addition of a new level (a 'split'). In practice this procedure may well save computation but the asymptotic performance is as before.

### ID4

If one considers ID4, the simplest of the true incremental versions of ID3, one can see that entropy calculations constitute a significant part of the tree generation process. This process consists of a number of operations :-

- growing branches
- adding examples to the tree
- discarding subtrees

The last operation can be disregarded as it can be considered to be constant and thus we concentrate on the first two. Tree growth consists of a number of simple steps (table 3.1). In the simplest case, tree growth will result in the addition of one further level below a node and can be characterised in terms of the above mentioned basic computational steps. This gives the following computation estimates for the operations listed in table 3.1. Part (1) would consist of $E_d$ string comparisons, where $E_d$ is the number of examples at the

node (level $d$) and is bounded by $E$. Part (2) consists of $A-d$ entropy calculations and is thus $O(A.V.C)$. Memory allocation in part (3) is assumed to be equivalent to the above basic mathematical operations and is thus $O(V)$. Part (4) consists of two steps: (a) up to $V$ string comparisons per example (finding the correct branch to take) and (b) $A-d-1$ statistical additions for each of $E_d$ examples (i.e. the *ICA*s described in §3.1.2). This gives part (4) an upper bound of $O(E(V+A))$ and thus we have an overall estimate of

$$O(E + A.V.C + V + E(V + A)) \approx O(E + A + E.A)$$

Part (5) is ignored as we are considering the addition of one level only. The second op-

| |
|---|
| 1. check if examples are one class |
| 2. calculate entropy over $A-d$ attributes |
| 3. attach up to $V$ children |
| 4. propagate $E_d$ examples down one level |
| 5. for each child, go to 1. |

Table 3.1: Tree Growth Steps

eration, adding examples to the tree, can be described as three basic processes, described here in terms of *one* example :-

- $V$ string comparisons on $A$ levels (find the correct child branch),
- $A-d$ ICAs on each of $A$ levels,
- $A-1$ entropy calculations (rechecking the test attribute).

These give estimates of $O(V.A), O(A^2)$ and $O(A^2.V.C)$ respectively. It is easy to see that $E$ exceeds the number of 'restructures' in any one run.

Thus we conclude that adding examples to the tree is the more computationally intensive operation and also, that entropy calculations are a major factor in algorithmic complexity for tree generation for ID4.

### ID5

For ID5, the process is more complicated; instead of being proportional to tree growth and example addition, tree restructuring is an independent process, computationally speaking[8] . In addition to the processes that ID4 executes, ID5 stores examples at nodes (largely implicitly) and reshapes subtrees instead of discarding them. Example-part storage is limited by the number of attributes per example and is thus $O(E.A)$. Restructuring consists of a number of sub-operations (table 3.2). Part (1) could be

| |
|---|
| 1. expand $V$ subtrees to include the 'pull-up' attribute |
| 2. pull new test attribute to root of each of the $V$ subtrees |
| 3. swap the current node with the new test attribute below |
| 4. update statistics (ICAs) for node and children |

Table 3.2: Tree Restructuring Steps

characterised as $V$ tree growths excluding the class comparison and entropy calculations[9] . The total number of decision nodes to be grown is bounded by the maximum number of leaves in the subtree, i.e. $V^{A-d}$, which is bounded by $V^A$. Each of these may require up to $E$ example propagations, for a cost of $O(E.A)$ (from before), to give an approximate estimate of $O(V^A.E.A)$. Part (2) is equivalent to $A-d-1$ invocations of parts (3) and (4) (i.e. it is a recursive process that 'pulls-up' one level at a time). Part (3) is again proportional to $V$. It can be approximated as the addition of one new node to the tree (the new test attribute) and then the addition of $V$ children for the old test attribute. This would give, from before $O(V)$ (i.e. no entropy calculations, class/value comparisons etc.). Part (4) is proportional to $E_d.A-d$. A simplistic view could be that it is equivalent to the usual ICAs but at one level only which gives $O(E.A)$ for the new subtree root and

[8] That is, for ID4, the rebuilding of a subtree is simply the same two operations over again, viz. tree growth and example addition.

[9] thus one requires steps 3 and 4 of table 3.1

the same for each of the $V$ children, i.e. $O(E.A.V)$. This gives an approximate total of

$$O(V^A E.A + A(V + E.A.V) + V + E.A.V) \approx O(E.A^2)$$

One can see that asymptotically the whole process is exponential in $V$ and quadratic in $A$. Thus we conclude that both entropy calculations and tree restructures are major factors of computational effort and can therefore have a significant effect on algorithmic efficiency.

It is difficult to assess the number of restructures which would be *required* in any one run in order to keep the most informative attributes at the higher levels of a tree. In the worst case, one would be required per example and would occur at the root[10] . Thus, in the worst case, the whole tree would need to be restructured, an estimate for which is given above (for one example). Therefore $E - 1$ restructures would give $O(E^2.A^2)$ calculations. The actual number and extent of restructures is dependent on the efficacy of attributes to describe the concept in hand and even the order in which examples are presented.

To summarise then, the number and cost of restructures for ID5 are bounded by $O(E)$, and $O(E.A^2)$, respectively. It is axiomatic that the number of *aborted* restructures, the computational cost of a restructure and tree size should be minimised to reduce effort. As for the impact of restructures on the need for further training, it is again hard to say.

## 3.3 Conclusion

> "Practical problems of induction involve two basic issues: efficacy and efficiency ...." (Rendell 1986)

We have shown how ID4 and ID5 can suffer with respect to computational efficiency and even efficacy, and we identify a number of causes. We conclude that entropy is not

[10] For ID5R it is conceivable that $A - 1$ restructures are performed per example.

an ideal purity function with respect to efficiency, that tree restructures are costly, are often unnecessarily executed, and may adversely affect efficacy and even the possibility of learning. We also comment on how these algorithms can behave in an uninformed, 'random' manner in searching the concept space, and the need for more guidance is apparent. We note that these problems occur with relatively simplistic algorithms, and may be compounded with more complex algorithms and data sets. We give some theoretical worst-case analysis and, in the sequel, we empirically evaluate these (and other) algorithms, and thus lend credence to our observations above. In the next chapter, we address these and other issues, as a major part of our thesis.

# Chapter 4

# JITTER

A fundamental hypothesis of this thesis is that present methods for the incremental induction of decision trees do more work than is necessary to produce a tree. We believe a more informed approach will enable an algorithm to reduce inherent computation whilst maintaining tree quality (e.g. accuracy, size etc.). Consequently we explicate our thoughts on improving this state of affairs. This chapter and those which follow, develop and present our original ideas, as a contribution to the state of the art in incremental decision tree induction.

## 4.1 Motivation

> "Learning effectiveness in complex domains requires the development of incremental, cost-effective methods." (Schlimmer et al. 1986c).

Many authors cite numerous characteristics of an ideal learning algorithm. In Rendell et al.'s (1987) words, one requires to make an algorithm *generally* more *efficient* and *effective*. Kearns (1990) identifies three general facets of PAC learning algorithms (§2.5.1), viz. learning *approximate* concepts through *general*, computationally *efficient* algorithms. Quick learning could be very beneficial in large and/or complex domains, i.e. increased

efficiency could increase tractability. Angluin et al. (1983) define a number of similar characteristics, i.e. a more *powerful* algorithm is one which infers more correct concept definitions; the *data efficiency* of an algorithm is determined by the number of examples required for it to converge to a correct hypothesis; and another measure of efficiency involves the number of times an algorithm *changes its concept description* (termed "mind changes") during learning, precisely what we identify as a major factor in ID5's computation estimates (i.e. restructures). However, according to Safavian et al. (1991), one cannot simultaneously optimise efficiency and accuracy. Often large amounts of time and storage are required and these may constrain tree type, numbers and types of features, and so on. All of these may prevent an algorithm from converging on a solution.

In an approach similar to ours, Utgoff (1995) notes: "The ability to restructure a decision tree efficiently enables a variety of approaches to decision tree induction that would otherwise be prohibitively expensive." An example would be, using a local, "indirect" method to build a tree (e.g. entropy) and then revising it according to a more global, "direct" method such as misclassification cost, size, and so on.

### 4.1.1 The Need for Speed

Schlimmer et al. (1986c) state that incremental systems suffer from the requirement that knowledge updates must be relatively speedy, otherwise one might as well restart the whole process with the new, enlarged training set. Incrementality in itself does not guarantee the "computational efficacy" of an algorithm, and thus speed is a prerequisite. Indeed, according to Weiss et al. (1991), most algorithms can classify quickly, and so one should concentrate on enabling an algorithm to *learn* quickly. Swain et al. (1977) use a concept evaluation function combining accuracy and execution time (see also Musick et al. 1993). Utgoff (1995) adds that the most important goal for an incremental algorithm is that its average incremental update cost (i.e. adding an example and updating the

tree) is less than the average cost of building the tree from scratch. Ideally one would like incremental algorithms to be as fast as, or faster than, non-incremental algorithms. Sutton et al. (1993) discuss "on-line learning", i.e. incremental, real-time learning:-

> "In a broad sense, on-line learning is essential if we want to obtain *learning* systems as opposed to merely *learned* ones."

Sutton et al. state on-line learning offers a number of advantages, e.g. it is potentially more robust (training set errors/omissions can be more easily 'corrected'), and training data can be generated relatively easily (via the performance system). On-line learning is necessary to track concept-drift, or "time varying functions". Kibler et al. (1988) note that as the number of attributes increases, learning rate (speed) decreases exponentially. See also Lebowitz (1987, 1988), Aha (1992), and so on.

### 4.1.2 Increasing Complexity and Size

> "Scaling up inductive methods to handle more complex inductive learning tasks remains a formidable challenge." (Shavlik et al. 1990)

Any algorithm to be used in the real world must be able to handle large, noisy databases in an efficient manner (Clark et al. 1989). Sleeman (1994) notes that KBSs are increasingly being integrated into conventional systems and adds:-

> "...if we understand how to formulate ML algorithms more simply, more *efficiently*, it will mean that *more demanding tasks can be solved* with a given amount of computing power." [Our emphasis]

Larger data sets allow the possibility of using more examples during training, but can thereby also imply increased run-times due to increased computation. Quinlan (1990c) suggests using larger example sets may be the only way to extract knowledge in some

noisy domains. Possible answers include a stronger bias, but this can affect generality and even tractability; fewer examples, but this ignores the advantages more examples can offer; or greater efficiency. Pre-processing may help, e.g. Gemello et al. (1989) use data reduction techniques to reduce the number of examples for learning. This is not ideal as the more examples that are used during learning, the more evidence there should be for a particular choice. Other techniques generalise examples into one, but this can lead to the loss of information and even over-generalisation.

Musick et al. (1993) refer to the use of very large sets as *mega-induction* (see also Cai et al. 1991), and propose the use of statistically generated subsamples for learning (cf. Quinlan's 'windows' §2.6.4). These authors attempt to control the trade-off between the *cost* of induction on larger sets and the increased *accuracy* they provide. However, this approach is still risky and adds computational overhead.

Some data sets are so complex (for instance chess) that often one must severely restrict the problem domain or the concept representation, i.e. bias (Utgoff 1986), or reformulate the problem in terms of fewer, more complex attributes in order to improve the tractability (see Shen 1992 for an example set which is intractable for ID5). Swain et al. (1977) note that as the number of attributes increases, accuracy will decrease, and recall Kibler et al. (1988), who state the learning rate decreases exponentially. Niblett (1987) notes that as the number of attributes increases, the sparsity of the example space will increase (for a given example set size) and so one must have larger example sets in order to learn accurate concepts. Sebag et al. (1991a) discuss an application domain in which an 'answer' is not known, either empirically or theoretically, i.e. true knowledge discovery occurs. Examples are expensive to represent and choosing ideal attributes is as complex as induction itself. The approach taken then, is to select some attributes, let the algorithm learn 'something' and then refine it. Thus an increased responsibility is placed upon a domain expert, something one should aim to reduce.

### 4.1.3 Increasing Informativity

Cestnik et al. (1987) discuss the 'evidence' afforded by larger data sets:

> "The confidence in a decision tree may depend on the number of instances in the leaves of the tree. .....if there is only one instance in a leaf, then the classification in this leaf is unreliable."

Fayyad et al. (1990) concur that the number of leaves is linked to error rate, i.e. reducing the number of leaves increases the example 'support' per leaf and so increases confidence in them. Thus any measure which allows the use of *more* examples in a given time, or makes better use of a *given* number of examples, consequently increasing the confidence in a tree, is to be encouraged.

### 4.1.4 Reducing Volatility

Fisher et al. (1989) note that early in training, many examples are inconsistent with the evolving concept description and that no one example should "irrevocably impact" this description. This is unlikely in an incremental setting[1] , but serves to suggest one should take care with what is changed, or indeed *not* changed, especially during early stages of learning. On the whole, it would appear one should instigate change when one has sufficient evidence that it is necessary. As we have shown, ID5 (etc.) adds (quasi-) backtracking to ID3, which can lead to oscillation of the concept definition. Decaestecker (1991) states that the risks of oscillation can be reduced by using the tree merge/split operators *only* when repeated calls are made, not immediately as per ID5.

Utgoff (1995) notes the "disturbing" variation in tree size during training, implying that frequent, substantial, 'root-local'[2] tree revisions are occurring. He concludes this

[1] due to back-tracking

[2] as opposed to 'leaf-local'

is due to the sensitivity of the attribute selection function (e.g. entropy) to underlying attribute-value-class frequencies. Thus it would seem prudent to gather more evidence before initiating change.

It is possible that IDL (Van de Velde 1989, 1990) could benefit from a more relaxed approach to transforming subtrees, especially as it often involves restructuring *and* pruning. The addition of another heuristic could help prevent IDL from limit-cycling through topologically minimal equivalent trees (e.g. the parity and multiplexor concepts, see Van de Velde 1989). It is noted however that Van de Velde's IDL will in fact reduce the number of actual and attempted restructures over ID5.

### 4.1.5 Increasing Generality

Other benefits of increased information pertain to the possible relaxation of bias. For instance, Seshu (1989) asserts that the pervasive Occam's Razor is not a panacea. If one can rely more on efficiency rather than bias then there is less domain knowledge to encode, less scope for incorrect bias, and consequently increased generality. Kibler et al. (1988) note how irrelevant knowledge can degrade performance. For instance, one could increase the complexity of decision functions by allowing linear and non-linear, multi-attribute combinations and constructive induction (e.g. Murthy et al. 1994, Utgoff et al. 1990). One could augment a representation's syntax, increase search 'fronts' (e.g. maintain multiple competing hypotheses), and generally postpone or avoid labour-saving techniques (e.g. search space pruning) until more evidence is available. For example, Clark et al. (1989) extend the search space of rules in CN2 (§2.6) by also considering inconsistent rules. Rendell et al. (1987) talk about dynamic bias, i.e. alterable during a run, and we believe this is a prerequisite for more general learning. Schlimmer (1987b) mentions representation change for avoiding language limitations.

Webb (1995) replaces heuristic searches with "admissible" ones (guaranteed to find

a target solution if one exists). Obviously this significantly increases computational loads and Webb utilises search space ordering and pruning to improve efficiency and tractability.

### 4.1.6 Conclusion

In summary, the potential benefits to be had from a more efficient algorithm are as follows:-

1. one can make use of larger example sets, and thus accumulate more evidence for any decision
2. one can process a given set of examples more quickly
3. one can make better use of current memory by using more attributes and/or information per example

and so the tractability of learning from example sets of increased size and/or complexity is increased (see also Musick et al. 1993).

Sufficiently fast incremental algorithms could become embedded in real-time knowledge based systems. Expert systems are often accused of being brittle and failing upon receipt of novel information. The addition of a fast learning element might at least ameliorate this condition and therefore the faster the better (e.g. Winkelbauer et al. 1991, Sutton et al. 1993).

Moreover, a better informed algorithm which relies less on user-supplied or domain specific knowledge will potentially:-

1. be more generally applicable
2. be less prone to inappropriate or wrongly specified bias

3. make fewer and/or better choices/changes

4. be able to adapt dynamically during training

5. be less 'volatile' and thus less likely to oscillate (Decaestecker 1991)

Thus we believe that one ultimate objective of AI should be real-time, incremental learning ("on-line", Sutton et al. 1993). Our goal then, is to induce a decision tree with maximal accuracy and clarity and minimal use of resources, in the shortest possible time. We also wish to accomplish this over the widest range of problem classes, with fewest assumptions, and least background knowledge. More specifically, we wish to build trees isomorphic with those of ID5, in less time, using less memory and less computation, without resorting to more stringent background knowledge.

Below we show how, using a straightforward heuristic approach, it is possible to achieve many of these aims. We show how one can decrease the work an algorithm must do to accomplish a task; how to increase the information available to an algorithm allowing more informed choice; how to reduce volatility and instability[3] during learning; how to decrease resource requirements; and how one could ultimately achieve greater algorithmic freedom (less bias). This combination of improvements represents our original work in this area.

## 4.2 The JITTER Family

Our main theme in what follows is computational efficiency. As we have shown, there are a number of aspects of incremental tree formation which can be reduced with a view to increasing the efficiency with which an algorithm works. These include the number of entropy calculations, the number and cost of actual and aborted tree restructures, the

[3] We define volatility to mean *frequency* of change and *stability* to mean the *extent* of a change.

impact on training of restructures, and the size of a tree. To reduce these quantities we believe we need a more knowledge-based approach[4] , but only through the use of 'non-parametric' information freely and easily available during induction. (Both these requirements, i.e. the use of freely available information and efficient learning are cited by Buntine (1990) as desirable.) We therefore preclude complex and/or time consuming calculations, (e.g. resubstitution as a means of judging tree accuracy), or domain dependent information such as user-specified background knowledge.

".... powerful induction is both model- and data-driven." (Rendell 1986)

Much like the 'smarter' versions of ID4 and ID5 which update on misclassifications, we aim to take our cue directly from the data set in use. Consequently we should not adversely affect the generality of our algorithm. For example, Cai et al. (1991) utilise a 'vote' per tuple in a database. As a tuple is generalised, its vote is passed onto its parent, and from these 'counts', two types of weight are generated, which allow the calculation of rule applicability/generality, discriminatory power, etc., and to prune the least useful ones.

One such source of information is tree quality. This could be used heuristically as a means of deciding when and what to restructure (for example) so that changes are no longer automatic, as under entropy. Thus we seek to augment the use of entropy (with explicit tree quality measures) when applied to the choice of *when and how much* to change, not *what to change to.* Norton (1989) makes such a use of decision tree quality (e.g. size, error rate, test cost etc.) as a means of guiding the attribute selection process. Arguably, tree *accuracy* is recognised as the foremost, or perhaps the most common criterion (Weiss et al. 1991) for judging induced concepts and so we concentrate upon it. Utgoff (1995) terms a similar approach "direct metric induction", where a measure

[4] because entropy is relatively 'blind'

of *tree* quality is used to form tests, instead of a measure of *test* quality. Van de Velde (1990) exploits knowledge directly from a tree's structure to change to a tree of optimal size, and examples focus such changes on 'needy' subtrees.

Below we initially present an intermediate algorithm, ID5_DELAY and then the JITTER (Just In Time Tree Evaluation and Recalculation) family. We show that it is possible to:-

- reduce run-time
- reduce the number and extent of restructures
- reduce the number of aborted restructures
- reduce the number of entropy calculations

whilst on the whole maintaining the accuracy of intermediate and final trees. Indeed, for some domains in which the ID3 family is known to have difficulties, JITTER actually increases the accuracy whilst reducing the complexity and the effort needed to produce the trees.

For clarity throughout this section we assume that each example is one of two classes (positive - the concept to be learned, or negative - some other concept), unless stated otherwise. However it is quite feasible, and indeed usual, for *all* algorithms reported here to manage more. We also assume, for simplicity, that attributes are categorical and that there are no missing values.

### 4.2.1 The JITTER Model

Essentially we use incremental TDIDT algorithms for a number of reasons. Such algorithms are geared to quick learning from relatively few examples. That is, a working hypothesis is very quickly constructed and then adjusted as further examples arrive.

Decision functions are '$V$-ary' (i.e. up to $V$ children) and thus no choice of a suitable partition (as for binary splits) is necessary, although this does have disadvantages for large $V$ or continuous values. Decision functions themselves are typically based on a single attribute, and once an attribute is used it cannot be re-used. Entropy and similar selection procedures help converge upon small trees, with least average depth (etc.), thus aiding tree update, and subsequent use. TDIDT algorithms therefore represent a class of algorithms geared to efficiency and learning performance. Quinlan (1990c) holds that trees are clear and concise, and are context sensitive (by virtue of 'independent' subtrees), and so are suited to our use of alternative bias, as shown below.

### Assumptions

A central assumption in our work is that extra, relevant information (bias or knowledge) can be gleaned from the data set. Such information is dynamic as it is alterable during a run, and is not fixed by a user. It is also automatic, as it is taken from the examples seen so far, i.e. one does not rely upon a domain expert. In essence this is nothing new as many authors use statistical formulae to augment decisions during learning (e.g. entropy, $\chi^2$). However, we do not use complex calculations, but prefer simple quantities, easily procured. Statistical tests can often make assumptions about the probability distribution of example sets, and/or become unreliable when small sample sizes are involved (e.g. $\chi^2$ for less than 5 values, Quinlan 1986b). Our techniques are designed to be general, and make no assumptions save that the data set is indicative of the target concept (something that is already assumed for induction). As such, no adverse constraints are required for the model to be realisable and applicable. In some sense our methods may appear ad hoc, but in fact we avoid making further assumptions about the applicability of statistical tests, contextual domain knowledge, meta-knowledge, expert perspicacity, and so on. We choose to illustrate these techniques through decision tree algorithms but see no reason

why they should only apply to trees.

### Model Quantities

In particular, our model utilises the following quantities :-

1. The number of examples seen so far (global)
2. The number of examples seen so far at a given node/leaf (local)
3. The number of entropy calculations
4. The number of restructures (local and global)
5. The number of aborted restructures
6. The number of levels in a tree
7. The number of node splits
8. Node purity (other than defined by entropy etc.)
9. Tree accuracy (global)
10. The number of nodes restructured

The relationships between these quantities are unclear but all depend to some extent on the first item. This embodies our central premiss, i.e. the use of the example set. The nature of the data can have a profound affect on all of these variables, e.g. noisy examples may directly increase items (3), (4), (5) (especially), (7) and thereby (6). Items (8) and (9) are also likely to decrease. It can be seen that (2) depends on both (1) and (7), as do (8) and (9). Increases in (3) are often concomitant with increases in (4). Such relationships are borne in mind during our investigations in Chapter 5.

In the absence of quantitative estimates of the *required* number of entropy calculations, restructures, etc., we settle for minimising such quantities, such that tree quality is not affected. In essence then, our model consists of using some of the above quantities to gather evidence about the quality of the current description and to base choices about changes/revisions on it. Thus our aim is to acquire what we call *data-specified knowledge.* Consequently we add further mechanisms to *aid* the evidential reasoning undertaken (e.g.

entropy), and also to exploit information from the tree under construction, as well as the example set. The above quantities can be thought of as a variation on the "multiple evaluation parameters", as depicted by Kononenko et al. (1991) and used to determine classifier performance *after* training (see also, end of §2.5.4. If such quantities are available *during* training, then they may well represent useful information for augmenting tree quality 'decisions'.

### Use of Accuracy

Safavian et al. (1991) comment on a tree pruning algorithm which works top-down *and* bottom-up. The tree is grown from the top down, but near the root one cannot know (without $K$-ply lookahead) what success one might have as growth continues. The provision of information from the bottom of the tree (where accuracy is best determined), can then influence this growing process. The full JITTER model (i.e. the JITTER and QUASI_JITTER algorithms, below) uses the aforementioned accuracy as a source of extra knowledge.

We conclude from $\widehat{\text{ID4}}$ (Schlimmer et al. 1986c) and $\widehat{\text{ID5}}$ (Utgoff 1988a) that the re-substitution error estimate of a tree should be a reasonable *guide* to overall tree accuracy. If not, the 'hat' versions of ID4/5 would be unreliable because they rely on *misclassifications* during learning. Therefore, we can use tree accuracy as a *guide* to the disparity between the actual and the desired tree. Note this does not take the place of entropy which we still use for attribute selection. Quinlan (1987a, 1990b) says:-

> "The accuracy of a leaf over the cases in the training set can be used to estimate the reliability of a classification arising from that leaf."

Quinlan gives a *pessimistic estimate* of error (more reliable with few numbers of examples) as $\frac{e-m-0.5}{e}$, for a leaf that classifies $e$ examples, $m$ of them wrongly. This implies one may

need to bias the use of accuracy information if it is based on resubstitution. HILLARY (Iba et al. 1988), uses an evaluation function to assist its hill-climbing which makes explicit use of *coverage* or loosely, accuracy. This is judged by keeping counts of the successes and failures of the current hypothesis to classify examples as they are added, i.e. provide an estimated error rate.

Utgoff's DMTI algorithm (1995) terms the use of a quantity such as accuracy as a "direct metric" (global) and allows different metrics to be used (e.g. tree size etc.). Thus we could extend our ideas in a similar way, to use different sources of information.

### 4.2.2 ID5_DELAY

```
For each example in training set
   While node not a leaf
      Go to correct branch
      Update attribute-value statistics
      If restructure_count = 0
         If attribute found with better entropy
            Increment restructure_count
      Else if restructure_count > 0 and < threshold
         Increment restructure_count
      Else
         If attribute found with better entropy
            Restructure subtree
            Reset subtree restructure_counts to 0
            Get next example and go to top
      Go to child node
   If leaf impure
      Grow tree
```

Table 4.1: Pseudo-code for ID5_DELAY

This algorithm[5] is slightly different to ID5 (there is also a version ID5_$\widehat{\text{DELAY}}$). It uses the same method of selecting a suitable test attribute from an example set, forming a tree, propagating an example down it and compiling statistics at each node.

[5] Based on an idea by an anonymous reviewer.

As per ID5, an example is propagated down a tree, according to the value of the current node's attribute. However, it does not always automatically recompute the entropy scores for each potential test attribute at every node the example passes through to see if restructuring is necessary. ID5_DELAY differs in that every node has an additional *counter*, initially set to zero to indicate that the node is *inactive*. As an example passes through the tree, if the current node is inactive, the usual entropy calculations are done to determine whether a tree change is needed. If a restructure is requested (i.e. another attribute has a better entropy score) then the node becomes *active* and the count is incremented.

At the start of a run, the user must set a global threshold which dictates how long a node must wait before restructuring is possible. For instance, one could set the threshold[6] to be '4'. A restructure is only allowed once a node's count is initiated and once that count has reached the global threshold. For each subsequent example which passes through an activated node, the count is incremented. Once the threshold is reached, the entropy calculations are redone to see if a better test attribute is still forthcoming. If one is found, the subtree is restructured as per ID5. Note that an inactive or 'subthreshold' node below a 'superthreshold' node is not prevented from being included in a subtree restructure if it is part of the subtree rooted at the superthreshold node. That is to say, the actual restructuring operation is exactly as for ID5. Once a restructure has occurred, the affected subtree node counters are reset to zero. These nodes will then once again stay inactive until another restructure is requested at them. Thus for an absolute threshold of 4, one can see that a maximum of two full entropy calculations can occur per set of four examples arriving at or passing through a node (on the first and fourth examples) instead of a possible four. It can also be seen that only one restructure will be allowed, instead of

[6] Thresholds can either be absolute values or percentages of the total number of examples seen so far.

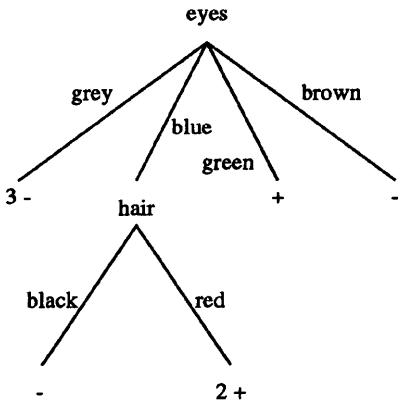
a possible four. Of course this does not imply that once a node becomes activated that a restructure must take place, the count may never reach the threshold, or the original test attribute may prove to be the best after all.

The aim is to avoid the effects of tree volatility until more examples exist at a node, the intuition, or heuristic being that a better decision should then be forthcoming (with respect to the efficacy of test attributes). This is analogous to the use of $\chi^2$ significance testing and other methods of pre-pruning of decision trees (Cestnik et al. 1987, Quinlan 1986a), which only allow tree growth once it is deemed to be advantageous. In our case however, basic tree growth is as per ID5, i.e. no pre-pruning or stopping criteria are used. Utgoff (1995) describes conditions for efficient incremental updates (§3.1.4). Use is also made of a "stale marker" at each decision node (remarkably similar to the counters in ID5_DELAY and PSEUDO_JITTER, ff.).

By way of an example, we extend the illustration shown in the last chapter regarding $\widehat{\text{ID5}}$ (fig. 3.4). Recall that $\widehat{\text{ID5}}$ has been presented with examples from an arbitrary extension of one of Quinlan's artificial example sets and that $\widehat{\text{ID5}}$ exhibits unnecessary tree changes. The three trees depict the situation initially, after the addition of one example, and after the addition of a second example. Figure 4.1 shows the same situation for ID5_$\widehat{\text{DELAY}}$ with a threshold of 10%. As can be seen, $\widehat{\text{ID5}}$ performs two changes, whereas ID5_$\widehat{\text{DELAY}}$ performs only one. The reduction in restructures has been counter-productive and not allowed a *beneficial* change, back to $\widehat{\text{ID5}}$'s final position. One should also note the additional entropy calculations and 'aborts'.

### 4.2.3 PSEUDO_JITTER

PSEUDO_JITTER goes one step further. The user must still set a global threshold at run-time but this time, for each node, the threshold represents the number of *requested restructures*. Thus for an absolute threshold of '4', PSEUDO_JITTER must attempt to




a) examples = 8

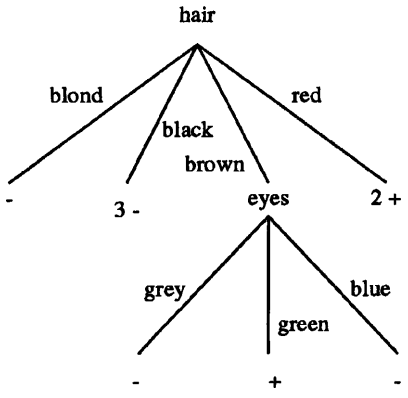
restructures = 0

nodes/leaves = 7

entropy calcs = 19

aborted restructures = 4

accuracy = 60%




b) examples = 9

restructures = 1

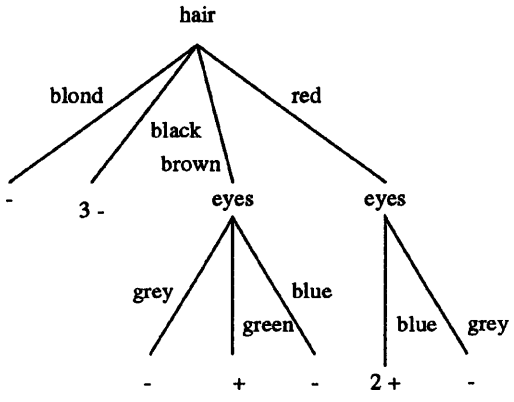
nodes/leaves = 8

entropy calcs = 22

aborted restructures = 4

accuracy = 40%

example no. 9 =
eyes=blue,hair=brown,height=tall,-




c) examples = 10

restructures = 1

nodes/leaves = 10

entropy calcs = 27

aborted restructures = 4

accuracy = 40%

example no. 10 =
eyes=grey,hair=red,height=tall,-

Figure 4.1: ID5$\widehat{\text{DELAY}}$ behaviour

restructure a subtree root four times before that subtree root is allowed to change. Once a restructure has occurred, the counters at each node of the subtree are again reset to zero. The heuristic used here is slightly different to that used before, i.e. a node's ability to restructure is more dependent on the usefulness of its test attribute (with respect to entropy) and less dependent on the number of examples at a node. That is to say, a node with sufficient restructure calls will be allowed to change, regardless of the number of examples at the node. Note that whilst many requests does imply a larger number of examples, there could be many examples and no requests.

```
For each example in training set
    While node not a leaf
        Go to correct branch
        Update attribute-value statistics
        Do entropy calculations
        If better attribute found
            Increment restructure_count
            If restructure_count >= threshold
                Restructure subtree
                Reset subtree restructure_counts to 0
                Get next example and go to top
        Go to child node
    If leaf impure
        Grow tree
```

Table 4.2: Pseudo-code for PSEUDO_JITTER

It is easy to see with both the above algorithms how one could cause the conditions for allowing a restructure to become unattainable (by setting the threshold too high, but see Chapter 6). The heuristic measures introduced so far still do not make explicit use of tree quality (i.e. accuracy) and thus still suffer from the disparity between such criteria and entropy (§3.2.1). The measures are simplistic and consequently do not satisfy our aim of making best use of information readily available during induction. PSEUDO_JITTER's response (with a threshold of 10%) to the same test set as shown in the previous section

is identical to ID5_$\widehat{\text{DELAY}}$ (including run statistics). Again, only one change occurs, such that accuracy has decreased and size increased, and so this particular threshold has *prevented* recovery (to $\widehat{\text{ID5}}$'s position).

### 4.2.4 QUASI_JITTER

```
For each example in training set
    While node not a leaf
        Go to correct branch
        Update attribute-value statistics
        Go to child node
    For each node leaf to root
        Add up majority and minority class totals
        Test ratio against global threshold
        If failure
            Do entropy calculations
            If better attribute found
                Restructure subtree
    If no restructure and leaf impure
        Grow tree
```

Table 4.3: Pseudo-code for QUASI_JITTER

For this algorithm, each example is propagated down the tree as before, but no entropy calculations are done. Once an example reaches a leaf, the algorithm retraces the path back up to the root node, stopping at each node, where the class purity is checked. The user sets a global threshold at the start of the run which indicates the level of accuracy *desired.* QUASI_JITTER must achieve this accuracy at each node in order to *avoid* an attempt at restructuring (analogous to Bhandaru et al. 1991 where node quality decides a possible 'split'). For example, with an absolute threshold of '2' (this indicates a *classification ratio* of 2:1, correct:incorrect), the majority class at a node must have at least twice as many examples as the minority class(es). If there are less, then the usual entropy calculations are performed to see if a better test attribute would be forthcoming.

One should note that tree update is now *multi-phasic.* That is, ID5 adds an example

and *simultaneously* checks to see if it sufficiently changes attribute-value counts to require a restructure, thus describing a *single phase* operation. The QUASI_JITTER algorithm splits this phase into two, viz. *example addition* and *restructure checking*. The result is that greater flexibility is afforded, i.e. one can choose whether to restructure from the top-down, the bottom-up, as an example is added, or afterwards. Indeed, Utgoff uses a similar idea (1995) to allow batch processing, i.e. *one* update procedure, after the addition of a number of examples.

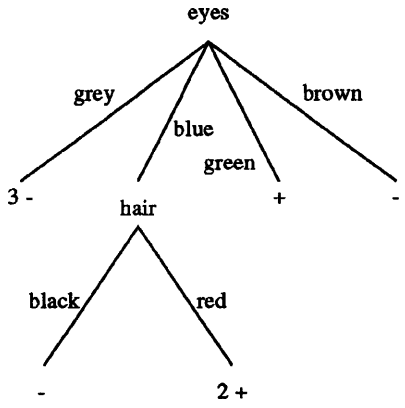
The heuristic notion here is that restructures are only necessary if tree quality (in our case, accuracy) becomes suboptimal. This begs the question of how does one reliably estimate a tree's accuracy whilst training, ideally without resorting to the overly optimistic, computationally intensive method of resubstitution? As tree accuracy is based on the classification of *test* examples at the leaves, a plausible assumption is that each leaf's accuracy is a useful constituent measure of global accuracy. Therefore, one could say that an effective tree path would result in a high proportion of one class of examples at any one leaf. Less effective test sequences (or perhaps a large percentage of noise) would result in a more even distribution of classes at a leaf. To a lesser extent, this should also be true of successively higher nodes in the tree. Thus our measure of accuracy at a node is judged by class distribution *whilst training*. If a node falls below a certain accuracy or purity, then we conclude that a better test attribute is required and a restructure should be attempted.

The threshold allows one to alter the required node purity and thereby the desired level of bias against misclassifications. One potential advantage with this approach is that restructure checking is performed bottom-up (as for IDL, §3.1.3 resulting in the possibility of causing less change overall. That is, we prefer to look for the least change to a tree (with respect to the number of nodes) by checking restructures bottom-up (leaf-local) instead of top-down (root-local) as in ID4 and ID5. The accuracy criterion is

easily extended to more than two classes by simply regarding all minority classes as the incorrect, or misclassified class.

For QUASI_JITTER, the accuracy criterion is based upon a node in isolation. It could be construed as being *in tune with* the gain in entropy from the level above. For instance, if the information gain is high, then the present node is *likely* to be relatively pure, whereas a small gain would imply a more even distribution of classes. Thus we have a vague information-theoretic measure without the need to recalculate entropy, something we are trying to avoid. However it does not make sense to use this accuracy criterion strictly at every node. As one nears a tree root, by definition there will be more examples from more classes. Thus we must reduce the accuracy requirement as we ascend. The method we adopt here is to reduce the accuracy criterion by a progressively larger proportion on each level as we near the root. The proportion is based on the number of levels and is calculated as "*current node depth / path length*". For example, a tree with a path consisting of five attributes will result in a reduction of the accuracy criterion of 20% for each level, up to the root. Figure 4.2 shows the previous example for QUASI_JITTER, with a threshold of 5:4, correct:incorrect. Note the reduced number of actual and aborted restructures, the much reduced entropy calculations (over ID5_$\widehat{\text{DELAY}}$ in figure 4.1, and consequently PSEUD$\widehat{\text{O}}$_JITTER), and especially the maintenance of accuracy for (b) and (c).

On the whole it is not obvious to us how to decrease the accuracy criterion in a natural and meaningful manner as we ascend, without resorting to a user-calculable parameter. However, as we have already noted, the accuracy calculated at a leaf could be a useful measure by which to judge each node on the path to the leaf. This leads us on to our final algorithm, JITTER proper.




a) examples = 8

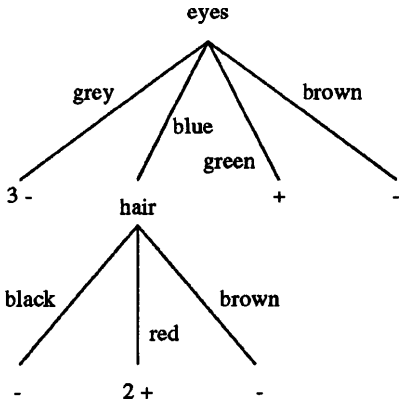
restructures = 0

nodes/leaves = 7

entropy calcs = 8

aborted restructures = 1

accuracy = 60%




b) examples = 9

restructures = 0

nodes/leaves = 8

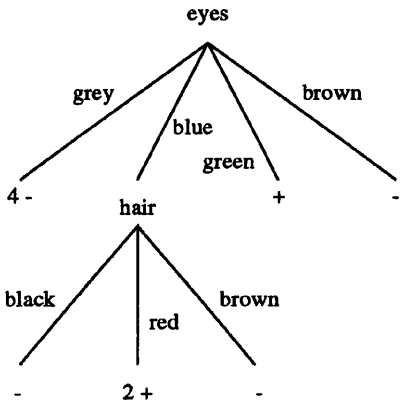
entropy calcs = 8

aborted restructures = 1

accuracy = 60%

example no. 9 =

eyes=blue,hair=brown,height=tall,-




c) examples = 10

restructures = 0

nodes/leaves = 8

entropy calcs = 8

aborted restructures = 1

accuracy = 60%

example no. 10 =

eyes=grey,hair=red,height=tall,-

Figure 4.2: QUASI_JITTER behaviour

### 4.2.5 JITTER

#### The Use of Weights

JITTER is again based on ID5 and so we concentrate on the extensions. JITTERś additions are not only extensions of the preceding ideas, they are also analogous to certain aspects of neural network research, specifically from the Back-propagation (BP) algorithm (§2.2.2). Others have shown how such synergism may pay dividends[7] , although we integrate ideas rather than full algorithms.

To this end, we augment our decision trees with *weights*, similar to neural networks. The use of weights in concept descriptions has been seen before, e.g. Salzberg (1990) and Schlimmer et al. (1986b), where weights are used in concept definitions to help the algorithm cope with noise and/or exceptions. See also Schlimmer (1987a), Aha (1992), Lebowitz (1987) and Rumelhart et al. (1986). In our case they are used in the induction process and do not appear in the final concept.

```
For each example in training set
   While node not a leaf
      Go to correct branch
      Update attribute-value statistics
      Go to child node
   Calculate certainty_factor and propagate to root
   For each node from leaf to root
      If weight <= 0
      Do entropy calculations
      If better attribute found
         Restructure subtree
         Reset weights in subtree
   If no restructure and leaf impure
      Grow tree
```

Table 4.4: Pseudo-code for JITTER

One is faced with a choice here, whether to attach weights to nodes or branches; we opt

[7] e.g. Sankar et al. 1991, Towell et al. 1992, Sethi 1990

for the latter to allow more flexibility. At each node is a list of potential test attributes and their respective values. Each tree *branch* has an associated weight which is used to indicate the efficacy of its parent test attribute with respect to the other 'potential' test attributes (and consequently the partitioning of the example set). An attribute's usefulness can differ between subtrees and levels. Weights attached to a node's branches allow a relatively fine-grained approach to attribute/value appraisal more in line with the synaptic weights of neural nets and the use of multiple weights in STAGGER (Schlimmer et al. 1986b), where each attribute-value pair is dually weighted. These weights will give a measure of previous successes and failures (with respect to example partitioning and consequently, accuracy), and are intended to reduce volatility and any tendency to oscillate between antithetical tree states during learning. The weights are set to zero as each node is created[8] so that all attributes start on an equal footing. Each attribute is thus in direct competition with each other attribute, for use as the test at a node (as is the case in entropy calculations).

Thus it would appear that weights are actually used to indicate the success of each individual attribute *value*. This is 'in tune' with the AID algorithm (Hawkins et al. 1982) which recognises that some attribute values may be less useful than others. We do not therefore, wish to amalgamate up to $V$ weights[9] into one at the parent node.

### Correction Factors

As an example is propagated down the tree, all that changes are the various statistical counts (ICAs) at each node (§3.1.1), and the entropy recalculations as seen in ID5 are omitted for the time being. One should again note that example addition is therefore *multi-phasic*, as for QUASI_JITTER.

[8] They are not random, as JITTER does not face the symmetry-breaking problems of neural nets.

[9] the maximum number of values

When an example reaches a leaf, a *correction factor* ($cf$) is generated based on the class distribution of the examples at the leaf, in terms of the class of the last example to arrive. That is, our *accuracy criterion* is based on *leaf purity.* Intuitively, the last example to arrive is more important (especially if tracking concept drift, Schlimmer et al. 1986a) and thus correction factors are couched in these terms.

For our accuracy criterion, we require a method which uses the information available locally, namely the examples at the leaf. This information should give a good indication of the effectiveness of the tests on the path to the leaf, and consequently, this path's contribution to global tree accuracy. Other more global information is available, such as the class distribution of the examples seen so far, the number of correct and incorrect classifications overall and so on. We feel that accuracy based solely on a *leaf's* class distribution is intuitively more sound than if it were based on an individual node at *any* level in a tree (as in QUASI_JITTER). One should conclude that in this case, the accuracy criterion is a measure of success of not only the leaf but all attribute tests leading to that leaf. Again we feel that this measure should vaguely reflect the information gain, but this time, from root to leaf. Thus we attempt to ascertain the efficacy of a sequence of tests, not one in isolation. This has similarities to COBWEB (Fisher 1987a, b), which implements polythetic as opposed to monothetic attribute selection and IDX (Norton 1989), which uses N-ply lookahead to the same end. So a 'bad' test sequence *should* instigate more changes, hopefully to rectify the situation. However, as always, it is left to entropy to choose the constituents of these attribute sequences.

Accuracy is also judged over a sequence of examples and not recalculated on each pass (as in QUASI_JITTER) without regard to what has occurred beforehand, through the use of the branch weights. Susceptibility to noise can be reduced by adding inertia to an induction method, thereby reducing the influence of individual examples (Schoenauer et al. 1990). Thus weights are a measure of the 'success' of attribute tests over time, *and*

a measure of 'success' of many attribute tests, both down and across a tree.

One might think that a heuristic based on a leaf's accuracy might be dangerous, i.e. it has the least example support in a path. However, it will at least be as pure as its parent, and is most likely to be *the* purest in a path. We do not see how nodes successively further up a tree can reliably predict the accuracy of a tree's leaves (and thus the whole tree). For instance, how can one use the root to determine the accuracy of a tree? Of course, the root is an important constituent, but its contribution cannot be judged in isolation, it must be judged *at* the leaves.

Such a local approach to $cf$ generation is simple, easy to implement and is more in keeping with the individual branch weights mentioned earlier. It is difficult to see how a more global approach could be related to individual tests and test sequences on the path to a leaf. Therefore, the best candidate for $cf$ generation appears to be the purity of a leaf as judged by its class distribution. Indeed no other local statistics are obvious to us. Intuitively then, one would feel that a pure or near pure leaf would indicate either a very dominant class globally or the tests en route are working as desired (of course a node's example count will have a bearing). The question is, how well are these tests working? Is it *necessary* to change them? Are there better ones available and what sort of trade-off can one expect between effort expended now and the accuracy/complexity of the final tree?

### Correction Factor Generation

A relatively pure leaf should give rise to increased confidence in its contributory attributes, while less pure leaves should result in a weakening (as per Back-prop). A leaf with similar numbers of positive and negative examples would be highly undesirable. If the original data set had a 50/50 class split, such a leaf would indicate no improvement at all. The problem is to find a (symmetrical) function which gives an equal magnitude

of certainty factor regardless of whether an example is correctly or incorrectly classified but which is still based on the class proportions. For a leaf example set of $\{+,+,-,+\}$ we would expect a lower score than for $\{+,+,+,-,+\}$, indicating that the greater the number of correct classifications, the more sure we can be of the effectiveness of the ancestor test attribute. Note that leaf impurity is not the same as impurity used in decision tree splitting functions, as here, {+,+,+,-} is not the same as {+,+,-,+}. In the first case the most relevant example (the last one), has been misclassified. An expression subtracting incorrect from correct totals has the unfortunate characteristic of giving least magnitude when there are similar numbers of either class, something that should be heavily penalised, and it is not a simple matter to subtract this value from say, 1.0. A simple and relatively continuous function consists of the ratio of current class count to the total of all classes at the leaf (similar to *static error* in Cestnik et al. 1987), i.e.

$$class\_count/total$$

where *class_count* is the number of examples at the leaf which are of the same class as the last to arrive and *total* is the number of examples, of all classes, at the leaf. (This is also reminiscent of the accuracy calculations depicted in Kononenko et al. (1991), but see end §2.5.4.) This has the desirable property of giving the greatest magnitude certainty factor (0.5) when there are equal numbers of each class and least when the leaf is purest (i.e. $cf \rightarrow 0$). A simple certainty factor (*cf*) generation algorithm is to prepend this real with a '+' or '-' according to whether the last example to arrive is in the majority (correct) or minority (incorrect) class(es). However, this function is not as unbiased as it at first might seem. It gives an intuitive score for correct classifications, i.e. the *cf* varies on the half open interval (0.5,1.0]. For misclassifications on the other hand, *cf* magnitude is a good deal less, varying [0.5,0.0). It is not hard to see how a few correct classifications can so load a set of weights that they take a disproportionate

number of misclassifications before a restructure is allowed. This scheme also reckons without pure nodes which give a score of 1.0, again over-emphasising correct classifications. The assumption that the prior probability of correct and incorrect classifications is 0.5 in both cases is obviously fallacious, otherwise entropy would be of no more use than random guesswork. If classifications and misclassifications give *cf*s of equal magnitude for corresponding class distributions, unless the misclassifications have a greater weight, an equal number of correct and incorrect examples arriving alternately will leave a weight unchanged. This is not desirable as it indicates no information at all has been gained. In such a case the search for better test attributes should be allowed a free hand. Thus in the absence of a more formal generation procedure we resort to a qualitative measure and bias our present function against misclassifications. That is, we place *more* weight on the effect of any misclassifications when deciding whether to allow change or not. This departure from our aim of relying solely on information gleaned from the data set, whilst ad-hoc, is not believed to be significant and we use a plausible value which is fixed for *all* tests. Indeed, it appears at present that this bias is an unavoidable consequence of using such a generation function. A bias factor of '1.0' is appropriate as it is equal to the theoretical maximum *cf* generated for pure nodes. Thus for a correct classification

$$cf \;=\; +\; class\_count \;/\; total$$

and for an incorrect classification

$$cf \;=\; -(\; 1.0 \;+\; class\_count \;/\; total \;)$$

Our *cf* generation function now varies as [-1.5,-1.0) and (0.5,1.0]. The above bias suitably counteracts the relatively high *cf* given for a pure node, and the relatively low probability of a misclassification occurring. Note that we do not allow a *cf* to be used for a leaf with one example. We feel a *cf* of '1.0' is too high a reward for such a sparsely populated leaf. Intuitively, this means that one should not allow either a weight increment or decrement

without some supporting evidence, i.e. other examples. Cestnik et al. (1987) state that one example at a leaf is unreliable with regard to classification (see also Quinlan 1993b). Lebowitz (1988) says such a notion is central to *deferred commitment.* Lebowitz is concerned with efficient learning, and to counteract problems caused by example ordering, UNIMEM can opt to suspend processing until more evidence is available on which to base decisions. Lack of evidence also motivates the use of *stopping criteria* (§2.7.5). For a problem with more than two classes, the majority class is the correct class, and all others are incorrect. If there is a tie, then the first class to arrive at the leaf is given precedence.

Once a *cf* is generated, it may be weighted by the proportion of examples at that leaf to the total number in the tree. Thus a sparsely populated leaf's volatility will be reduced and therefore its contribution to a relatively popular ancestor. In other words, infrequent misclassifications associated with statistical 'outliers' will have less chance of causing wholesale restructuring of the tree.

To illustrate the above ideas, consider a leaf with six positive and four negative examples. The correction factors are then as follows. If the last example to arrive was negative then

$$cf = -(\,1 + (\,4\,/\,(\,6 + 4\,)\,)\,) = -1.4$$

otherwise,

$$cf = 0.6$$

### Weight Update

The *cf* is now used to update the weights on the path from the leaf to the root. It should be apparent that *cf*s do not always decrement a weight. In fact, if the example is correctly classified at the leaf, then a positive factor is propagated which will increase the weights on the path to the root. Only when a misclassification occurs at a leaf are the weights

decremented. This is somewhat similar to the 'hat' versions of ID4 and ID5 which focus on misclassifications (i.e. only allow a restructure if an incorrect classification occurs). In our case however, *correct* classifications are not ignored and still play a significant part in any decision to restructure.

The correction factor may be decreased by a fixed percentage at each level. The reason for the decrease is so that correction factors do not have such a pronounced effect on the nodes further up the tree. This is in keeping with the observation earlier for QUASI_JITTER that the accuracy criterion should be relaxed as one progresses up the tree. The intention is that the algorithm should attempt to rectify any misclassifications by restructuring the smallest subtree possible and only if this is unsuccessful will larger subtrees be changed. Thus correction factors will have the largest effect on nodes closest to the misclassifications (i.e. the leaves). In practice, $cf$ demotion was found to be unnecessary and as its use would constitute a free parameter, we do not use it here.

Weights are updated using a formula which takes account of previous successes and failures and the current *cf*. A simple and intuitive method is similar to that used by Back-prop (§2.2.2):-

$$w_i(t+1) = w_i(t) + cf$$

where $w_i(t+1)$ is weight $w_i$ at time $t+1$, similarly for $w_i(t)$, and $cf$ is the correction factor.

### Tree Checking

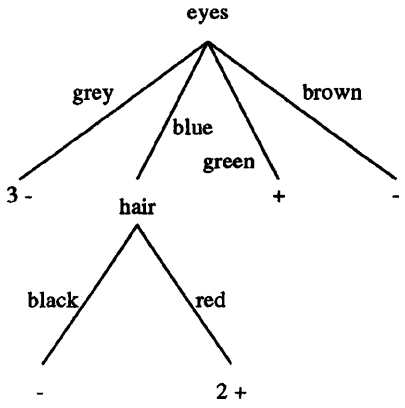
The weights along the 'active' path are now checked against their threshold, namely zero, starting with the leaf and moving up to the root. Theoretically, these weights can have any value between $-\infty$ and $+\infty$. If a weight is found to be less than or equal to zero (its start value) then the restructuring phase commences. The entropy calculations which are always done in ID4/5 are now done to re-establish the best test attribute. If the

current attribute is still the best then this phase ends, otherwise the tree is restructured as for ID5. If a subtree is changed, then all weights are reset to zero, as when each node was first created. This is done to let the subtree 'start afresh' and assumes that the restructure has been worthwhile i.e. there are fewer 'bad' test attributes and therefore fewer misclassifications. The process continues at the next level up until the root node is reached. The overall *cf* propagation process is analogous to the Back-prop algorithm in that an error 'distance' is computed at the output stage and is used to update the weights back through the 'network'.

### Example

Here we give a simple illustration of where JITTER can help induction by reducing tree volatility and simply waiting for more 'evidence' before restructuring. Extending the earlier example, JITTERś response to the same set can be seen in figure 4.3. Figure 4.3(b) shows JITTERś tree after one addition. The tree structure has changed only very slightly (addition of one node) and the accuracy has remained at 60.0%. We believe JITTERś tree to be superior (to $\widehat{\text{ID5}}$) in this case due to the retention of the prior accuracy and only minor increase in size. This also has the indirect benefit of less computation (i.e. no restructure). Part (c) shows the benefit of waiting, as $\widehat{\text{ID5}}$ now changes back to this tree. Note the differences in statistics for each algorithm, especially the *further* reduced entropy calculations and 'aborts' as compared to QUASI_JITTER (fig. 4.2). It should also be apparent that $\widehat{\text{ID5}}$'s restructure could equally well improve accuracy and decrease complexity. This is precisely our aim, changes should be done when, and only when, necessary.

Again it should be apparent that JITTER restructures from the bottom-up, with the resulting potential for its restructures to cause less change overall. Our algorithm should appeal to advocates of Marr's Principle of Least Commitment (Marr 1982) in




a) examples = 8

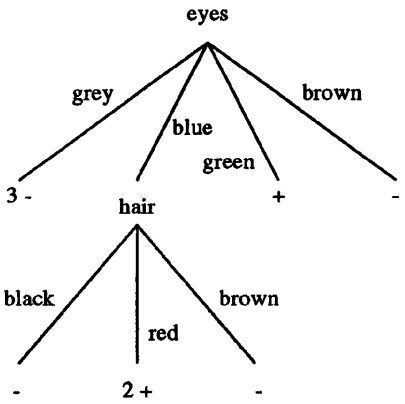
restructures = 0

nodes/leaves = 7

entropy calcs = 5

aborted restructures = 0

accuracy = 60%




b) examples = 9

restructures = 0

nodes/leaves = 8

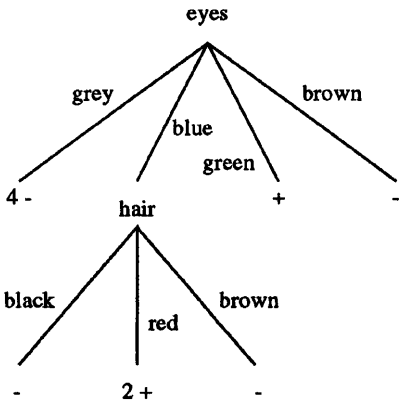
entropy calcs = 5

aborted restructures = 0

accuracy = 60%

example no. 9 =

eyes=blue,hair=brown,height=tall,-




c) examples = 10

restructures = 0

nodes/leaves = 8

entropy calcs = 5

aborted restructures = 0

accuracy = 60%

example no. 10 =

eyes=grey,hair=red,height=tall,-

Figure 4.3: JITTER behaviour

that it tries to avoid tree changes until there is a greater level of certainty that one is required. It then prefers 'leaf-local' to 'root-local' restructures and only if these do not suffice will it consider changes nearer the root. Marr goes on to say that one should avoid hypothesise-and-test algorithms. It can be seen that our algorithm reduces the hypothesise-and-test element of ID5 by reducing entropy calculations and the number and extent of restructures (see next chapter). The intention is to go beyond the mere generate-and-test nature of entropy and to aid the selection procedure with a more informed, knowledge-based (albeit simplistic) method.

### Variations

It should be noted that JITTER is at present an extensive program with the ability to test a number of versions involving the generation of correction factors, weight updates, threshold testing and so on.

Here we illustrate some variations of the JITTER algorithm. Conroy et al. (1994, 1995) report on the testing of JITTER with a number of training sets (see also Chapter 5). In one set in particular, it is noted that as the percentage of noise is increased, the level of tree manipulation activity also increases. One could say that if the attributes are not sufficient to discriminate between noisy examples, a tree induction algorithm could find itself cycling between subtrees which have different constituent test attributes. Van de Velde (1989) describes the phenomenon as 'limit-cycling', as IDL cannot unequivocally decide between a number of trees. Similarly, Decaestecker (1991) notes that incremental procedures can cause oscillation. Van de Velde (1990) notes that given an unlimited supply of examples, ID5 does not stabilise to a final tree for the multiplexor concept. As entropy is largely a 'single-ply' procedure, concepts which require more than one attribute in sequence to discriminate effectively between examples can cause untold havoc on the efficiency of an algorithm. Thus one could assume (as we show below) that many *aborted*

restructures could indicate an inability to find a better generalisation and thus indicate the presence of noise.

Specifically, the number of actual and aborted restructures increases significantly. In order to make use of this salient information, we add a weighting factor to the *cf* as it is generated at the leaf. This factor involves the number of global, *actual* and *aborted* tree restructures and is calculated as follows :-

$$w = \textit{no. aborted restructures} \; / \; \textit{no. restructures}$$

It is used thus :-

$$IF \; \textit{no. aborted restructures} >= \textit{no. restructures} \; THEN \; cf = cf + w$$

Therefore as the number of aborted restructures increases in proportion to the number of actual restructures, we conclude that noise is playing a significant part in the tree manipulation history and the node weights should be increased sufficiently to preclude further unnecessary change. This stems from the observation that consecutive misclassifications will result in the decreasing of the weights along a particular path. Thus JITTER will attempt to restructure that path once the weights are less than or equal to zero. If the current test attributes are in fact the best that are available with respect to entropy, then this restructure is aborted. Therefore, we should have left the test attribute sequence alone and not wasted time and effort trying to change it.

Similarly, a relatively large number of *actual* restructures (as compared to aborted restructures), might indicate that we are experiencing a cyclic restructuring pattern as observed with the parity and multiplexor concepts. Again, restructures can then be decreased to avoid wasted effort. In a comparable way, too few restructures might lead to overly complex trees and thus one should decrease the node weights to allow more. This is due to the fact that if little or no restructures are allowed, JITTER tends to rely on the splitting function to separate examples of different classes (see Chapter 6).

In (Conroy et al. 1995) we use another, simpler *cf* generation procedure, namely *majority class / total* and prepend this with a plus or minus according to whether the last example was correctly or incorrectly classified. From before, if the last example to arrive was negative then

$$cf = -( 6 / ( 6 + 4 ) ) = -0.6$$

otherwise,

$$cf = 0.6$$

Another alternative is actually to allow *top-down* restructure checking such that *one* restructure is allowed, i.e. we resort to ID5's root-local preference. The results are shown in the next chapter.

A number of other variations were subject to experiment, but have little or no benefit, but we list them here for completeness. Fixed term *cf*s do not appear to work at all well, and would also constitute a free parameter. Weights may be attached to *nodes* as opposed to branches, but this appears to detract from the current branch method. It is possible to check an attribute's weight against all *others* at a node, as opposed to zero. However this does not make sense unless one updates the *potential* test attributes at a node as well as the current test. Thus one must decide to which attribute a *cf* applies, which adds complexity.

## 4.3 Worst-Case Analysis

### 4.3.1 ID5_DELAY

ID5_DELAY performs the same basic tree operations as ID5 but with the small additional cost of keeping a counter at each node. Thus for any threshold setting, we potentially have an extra addition and comparison for each node through which an example passes.

If a subtree is restructured, the counts are reset. These two operations give estimates of $O(E.A)$ and $O(E.\sum_{i=0}^{A} V^i)$ basic operations overall, respectively, whereas an ICA estimate is identical to ID5.

ID5_DELAY(0)[10] behaves in a similar way to ID5 except for a slight reduction in restructures etc. due to the algorithm always requiring at least one *further* example at a node before it will check the threshold. That is, a restructure is *requested*, then the *next* example allows a possible change[11] . For ID5_DELAY(100) there will be no restructures at all (see below). Thus ID5_DELAY(0) would most likely recalculate entropy $O(E.A)$ times (test attribute checking) in adding $E$ examples, whilst ID5_DELAY(100) would give 0. Estimates of computational effort will therefore be reduced accordingly. Lower thresholds will give lower reductions, though not necessarily pro rata.

For restructuring, the worst case of one per example which applies to ID5 will also occur for ID5_DELAY. Due to the fact that one requires a minimum of two differently classed examples before a tree is formed and thus before a restructure can take place, the 'counter' will always be at least two less than the example count. However, the restructure count for ID5_DELAY(0) will tend to $E$ as $E$ becomes large. The restructure count for ID5_DELAY(100) is guaranteed to be zero as the counter can never 'catch-up' with the example count. Again, restructure counts for thresholds between these extrema are *likely* to be reduced over ID5. To date, we know of no method of ascertaining the *expected* restructure count quantitatively and believe one will not be forthcoming until one can reliably predict where and when a restructure will occur. Thus we conclude that in the worst case, ID5_DELAY will perform slightly more operations per example to give a larger computation estimate than ID5 (depending on the threshold, but assuming one precludes the 100% case which simply stops all restructures). However, there is potential

[10] That is, ID5_DELAY with a threshold of 0, i.e. 0% of the global example count.

[11] Note that a threshold of 1 is effectively the same as one of 0.

for savings in the number of entropy calculations and restructures in the average case.

### 4.3.2 PSEUDO_JITTER

For PSEUDO_JITTER(0), one will see behaviour very similar to ID5 with the slight added cost of counter maintenance. However PSEUDO_JITTER(100) and threshold values in between will not reduce the number of entropy calculations. The ID5-style calculations must still be repeated at each node through which an example passes, in order to update the 'restructure' counts therein, giving an estimate of $O(E.A)$, as for ID5. On the other hand, one should see a reduction in the number of restructures. For a restructure to be allowed for PSEUDO_JITTER(100), one restructure request must be made for *every* example added to the tree and that request must be made at a node through which *all* examples have passed, i.e. the root. Thus the number of restructure requests must be equal to $E$ and this is not possible (see above). Thus PSEUDO_JITTER(100) will result in zero restructures, and computational effort will be reduced accordingly. We note that PSEUDO_JITTER, in the worst case, will do more work than ID5, but that there is scope for a reduction in the number of restructures.

### 4.3.3 QUASI_JITTER

QUASI_JITTER behaves quite differently to the above algorithms. There are no counters to maintain and no automatic entropy calculations at each node through which an example passes. Thus for adding *one* example to a tree, we have $O(V.A + A^2)$ basic operations as opposed to ID5's $O(V.A + A^2 + A^2.V.C)$. However, ID5's estimate includes the tree 'checking' process of recalculating entropy. QUASI_JITTER's next task of checking each node on the current example path for its accuracy as opposed to its entropy, gives a similar but smaller estimate of $O(V.C)$ basic operations per node, per example. Thus

we have $O(A.V.C)$ for QUASI_JITTER, and $O(A^2.V.C)$ for the corresponding part of ID5. For each restructure, QUASI_JITTER would need to recalculate entropy to find the best attribute, whereas ID5 will have already ascertained the best one during its tree checking. Thus in the worst case, again for one example, if entropy were recalculated at every node on the current path, we arrive at $O(V.A + A^2 + A.V.C + A^2.V.C)$ basic operations for QUASI_JITTER assuming no restructures, a slight increase over ID5.

As one cannot quantitatively estimate where a restructure will take place, it is sufficient to say that both algorithms will have the same estimate of computational effort for restructures, but that it is likely that QUASI_JITTER will cause less change. This is because it engages in checking and therefore effecting, restructures from the bottom up. Thus if we assume that all nodes in the current path from root to leaf fail the entropy or accuracy tests, then ID5 will restructure the whole tree whilst QUASI_JITTER will restructure the subtree rooted immediately above the leaf. However this ignores the fact that the computation required for ID5 to add an example to the tree, will be curtailed once a restructure is initiated. Thus one should ask whether the restructuring of the whole tree is more or less computationally expensive than the addition of one example to the tree, the checking of the accuracy criterion and the restructuring of a minimal subtree of $O(V)$ nodes. It is, however, quite possible for ID5 to cost less if we assume that both entropy and our accuracy criterion result in the same restructure (i.e. the same level and attribute in identical trees), especially if it is nearer to the root. For example, if a restructure occurs at the root then ID5 will begin by adding the example to the root node, checking entropy and then restructuring the whole tree. QUASI_JITTER must propagate the example to a leaf, check all node accuracies on the path to the root (perhaps recalculating entropy for each on the way), and *then* restructure the whole tree.

QUASI_JITTER(0) and QUASI_JITTER(1) (absolute thresholds) will on the whole behave the same. Here the accuracy criterion is such that it will accept any mix of

classes at a node and thus avoid *any* restructures. QUASI_JITTER with some arbitrarily maximum threshold will attempt to restructure everything, thus increasing the tally of aborted restructures and not saving any actual restructures or entropy calculations over ID5. It is easy to see how relatively small changes in the threshold can drastically alter the behaviour. Therefore we conclude that QUASI_JITTER could cost considerably more to construct a tree than ID5, but that there is considerable scope for reductions in the number of entropy calculations and the number and extent of restructures. Estimates for ICAs however will be the same as for ID5.

### 4.3.4 JITTER

JITTER is able to replace $O(A^2.V.C)$ entropy recalculations (test attribute checks) with one addition (weight update) and one comparison (weight checking) on many occasions. As for QUASI_JITTER, when adding new examples to a tree, JITTER does not recalculate entropy. It does however calculate a *cf*, propagate it to the root and then check the path, leaf to root for a zero or subzero weight. Any such weights will result in entropy recalculation and a possible restructure. Thus in adding one example to a tree, JITTER will compare attribute values and perform ICAs only, to give an estimate of $O(V.A + A^2)$ basic operations. The cost of *cf* generation depends on $V$ and $C$, i.e. the instance-counts at a leaf, and is thus bounded by $O(V.C)$. The cost of *cf* propagation depends on the depth of the tree (a path length of $A$ is assumed) and gives a cost of $O(A)$. Weight checking also gives a computational cost of $O(A)$ basic operations, again per example. At a similar stage, ID5 would have executed $O(V.A + A^2 + A^2.V.C)$ basic operations to JITTER's $O(V.A + A^2 + V.C + A)$, per example. However, this assumes that no weights are in fact less than or equal to zero. In the worst case then, entropy will be recalculated at every node on the path to the root for a slight increase in overall cost, i.e. $O(V.A + A^2 + V.C + A + A^2.V.C)$.

As for restructures, JITTER performs bottom-up attribute checking and so similar observations apply here as for QUASI_JITTER. In the case where both algorithms restructure the same tree, JITTER will again give a slightly higher computation estimate which includes the resetting of the subtree weights (bounded by the maximum number of nodes/leaves in a tree, i.e. $\sum_{i=0}^{A} V^i$). Empirically, JITTER will usually restructure fewer nodes and thus give a lower computation estimate than ID5, but as mentioned above, we have no way of proving this quantitatively. We conclude that it is likely that JITTER will expend less effort in adding examples to a tree, and will restructure fewer nodes and less often compared to ID5. However, it is possible that JITTER will perform the same number of entropy calculations and restructures for a slight increase in computational effort.

## 4.4 Summary

We suggest there are numerous reasons for increasing the efficiency and the knowledge-based guidance of an algorithm. This allows greater flexibility and power, and less need for bias during induction. We show how one can augment tree manipulation decisions with information gleaned from the example set, and thus how to exploit tree quality and the example set more fully. We show how this might be used to increase efficiency. We design a rudimentary model of (data-specified) knowledge-based induction which forms part of the basis for our investigations. We identify and design four algorithms, based on ID5, which implement some of these ideas, and with which we test our hypotheses. These algorithms are described in detail. ID5_DELAY is based on the simple idea that the average number of restructures per example is probably too high and can be reduced. PSEUDO_JITTER is based on the simple idea that the average number of restructures is probably too high and can be reduced. Whereas QUASI_JITTER and JITTER are based

on the simple idea that restructures are only necessary if tree quality decreases. Some or all of these algorithms may decrease concept volatility, increase stability, decrease susceptibility to any one example, improve flexibility, use dynamic bias, adhere more closely to the Principle of Least Commitment, and make the use of tree quality more explicit. Our worst-case analyses hint at extra complexity to achieve these aims, but are insufficiently fine-grained to be considered definitive. On balance, we believe these algorithms will significantly reduce inductive effort and therefore we turn to empirical evaluation to give some measure of average-case performance. Here we find that our hypotheses do indeed significantly improve inductive efficiency.

# Chapter 5

# Experimental Analysis

In this chapter we empirically evaluate our and others' algorithms with respect to the efficiency of induction. We aim to improve performance through maintaining or improving complexity and accuracy, whilst reducing run-time, the number of entropy calculations and the number and extent of actual and aborted restructures. In particular, the following points will be brought out in the detailed examination which follows:-

- $\widehat{\text{ID5}}$ can waste significant effort during induction
- JITTER and QUASI_JITTER are able to achieve excellent run-time, complexity, accuracy and efficiency scores, and also reduce volatility and increase stability
- JITTER's and QUASI_JITTER's increased efficiency often makes them more comparable to the non-incremental ID3
- JITTER and QUASI_JITTER are able to save on both time and memory and so increase the tractability of some sets, thus producing an 'answer' where $\widehat{\text{ID5}}$ could not

We start with a brief illustration of the relative merits of the existing ID3 family before progressing on to the JITTER family, where we compare various versions. Subsequently

we test the full JITTER implementation further and compare against ID3 and $\widehat{\text{ID5}}$[1] . Finally we show the effects of noise and pruning.

## 5.1 Procedure

The results shown below are from our own implementations of ID3, ID4, ID5 and their variations. Whilst we have attempted to ensure each is as close to the original as possible, we accept full responsibility for any discrepancies and thus any anomalies in results as a consequence. **It should be noted that the JITTER family is developed from our ID5 and so any changes from the original ID5 will also apply to our algorithms, thus helping to ensure the validity of our comparisons.**

In the following, each data set is randomly divided into a training set (two thirds) and a testing set (one third). This is repeated to give ten different, randomly selected training and testing set pairs and the results are then averages over the test sets. In doing this we seek to minimise the effects of example ordering and thus ascertain more realistic true error rate estimates.

Whenever practicable, we use the "t-test for small sample distributions" to determine statistically the level of significance which can be attached to test means. Figures given in '$x$%' form indicate a confidence level of $x$ per cent that the means we compare are significantly different (see also Appendix A.3).

Some example sets may have a certain percentage of *noise*, e.g. a noise level of 10%, which means a random ten percent of the examples have had one *attribute value* randomly altered.

The LED set for example, also classifies examples into one of *ten* classes, whereas less complex tasks might typically use only two. Such facets are necessarily dealt with in any

[1] Thus one can glean *some* comparative information between say, JITTER and ID5R by generalising from the first illustrations below.

algorithm with 'real-world' capabilities. Time has precluded our adding protocols to deal *intelligently* with continuous and/or missing values, to *any* of the algorithms mentioned. We do not believe this affects the validity of any evaluations and/or conclusions, but does represent some inconvenience with regard to usable test sets. Moreover, as mentioned in previous sections, dealing with continuous and missing values implies intensive calculations, and any increased efficiency, parsimony, etc. shown by our algorithms should manifest *further* increases in such situations.

It should also be noted that we add one example at a time to a tree. It is generally necessary to do so with all algorithms used, i.e. each example, whether *accepted* in blocks or not, is *presented* to a tree individually. The only exception is ID3, i.e. it searches all examples for the best split, in toto. This *single submission*[2] protocol also allows one to observe exactly what occurs at each stage of a tree's development, i.e. calculate *intermediate* accuracy, complexity, effort, etc., after each example is added.

One should also note that our values for run-time are less 'dependable' than other more 'deterministic' quantities. Whilst we utilise *cpu* time, as opposed to *elapsed* time to record tree construction durations, one is always at the mercy of the network in multi-user, multi-tasking systems. However, where possible, a minimum run-time, corroborated by others, is recorded.

*Empirical* analyses are somewhat limited as one cannot assume that results from specific problems indicate performance across all domains, or even on general *classes* of problems. Heuristic search is difficult to analyse and so empirically, one desires 'representative' example sets. Consequently we attempt to test all algorithms concerned with as many example sets from as many diverse domains as is practicable, in the hope that one may draw (quasi-) general conclusions. It is axiomatic then, that such an methodology

[2] As opposed to *block submission*.

is intrinsically falsity-preserving[3] .

## 5.2 Test Domains

Here we give a general overview of the test domains used and what they may demonstrate. For full details see Appendix B and/or (Murphy et al. 1994a).

### 5.2.1 Balloons

We use this domain to show performance on simple concepts. ID3 etc. should have few problems with these concepts, i.e. there is no noise, no missing values, few attributes, and so on. There are four small data sets concerning the inflation of balloons. Each set has sixteen examples, with four binary valued attributes. The last set corresponds to a simple DNF problem.

### 5.2.2 Chess

#### KRKPA7

This set depicts the chess end-game of *king and rook versus king and pawn on A7*, abbreviated to *KRKPA7*. There are some 3196 examples each of which has 36 attributes and are classified according to whether white (king and rook side) can win or not, with white to move (Shapiro 1987, Muggleton 1987). We use this set as it is considerably more complex than many here with regard to the number of attributes and examples. Consequently there is considerable scope for loss of efficiency.

[3] As perhaps befits induction!

#### Knight Pin Chess End Game

In this set there are 1000 examples each with 22 attributes, and no noise or missing values. The class is decided according to whether the knight's side (black) loses in $n$-ply, where $n = 2$. The pieces consist of a king and knight (black), plus a king and rook (white).

### 5.2.3 Breast Cancer

This breast cancer data was originally provided by Zwitter et al.. There are 286 examples, each of 9 attributes, some of which are linear, the rest categorical, and 2 classes. There are some missing values, which for our purposes have been removed (i.e. the examples are ignored[4] ). This set gives a good example of the real-world data with which a learning algorithm must cope, i.e. multi-valued attributes that are noisy, possibly inconsistent and inherently inadequate for depicting the (poorly understood) underlying concept(s) (i.e. high residual variation (Mingers 1989)). The linear attributes may cause excessive fragmentation of the example set (due to entropy) and there are relatively few examples for such a complex domain.

### 5.2.4 Cardio-Vascular

This set resulted from research carried out at the Renal Unit of Manchester Royal Infirmary and involved *Vascular Disease in Uraemia.* The study was intended to predict the likelihood of death from heart failure within four years of a kidney transplant. There are 3 classes and 27 attributes (e.g. age, sex, angina, diabetic, smoker). Again, this set represents a complex real-world problem.

[4] There are only nine examples with missing values.

### 5.2.5 LED

Next we look at the 7 segment LED display as mentioned by Breiman et al. (1984). This concept has 1000 examples, 10 classes ('0'-'9') and 7 Boolean attributes (indicating whether each LED segment is ON or OFF). Noise levels range from 10% (i.e. each attribute has a 10% chance of being inverted), to 50%. Quinlan (1987b) notes that the LED sets (Breiman et al. 1984) are useful for testing the ability of algorithms to deal with noise and complexity.

### 5.2.6 Lenses

This set depicts the fitting of contact lenses. There are 24 examples, each with 4 nominal attributes, and 3 classes corresponding to the fitting of *hard, soft*, or *no* contact lenses. The set is complete, with no noise or missing values. Some 9 'rules' cover the training set, which together with the small number of examples makes this a difficult domain in which to describe accurately the underlying concept(s).

### 5.2.7 Lung Cancer

This set describes 3 types of lung cancer, with 32 examples of 56 nominal attributes. There are 5 missing values, which again necessitates the removal of the offending examples. No other information is given on attribute descriptions (Murphy et al. 1994a). The large number of attributes and few examples in such a complex domain makes accurate classification difficult.

### 5.2.8 Lymphography

This lymphography data was originally provided by Zwitter et al.. There are 148 examples of 18 attributes each, no missing values and 4 unevenly distributed classes. Again

this constitutes a real-life domain of considerable complexity.

### 5.2.9 Mux

Our next test set is the 11-bit multiplexor with 2048 examples, 8 data bits, 3 address bits and a class bit. The mux is difficult because the relevance of the data bits is a function of the address bits. Thus the class is a function of address and data bits, whereas one normally assumes attributes are independent of one another. Quinlan (1988b) notes that the C4 algorithm tended to 'start' a tree with a data bit. This is a difficult set (especially with added noise) for any monothetic (§2.7.2 and also §5.2.11) classifier.

### 5.2.10 Noughts and Crosses

This set depicts a 'Tic-Tac-Toe' end-game. The examples describe all possible end-game configurations where 'x' is the first to play. There are some 958 examples, each with 9 ternary attributes (one square on the board) and no missing values. It is noted (Murphy et al. 1994a) that this set gives ID3 "fits".

### 5.2.11 Parity

The next set is the *even parity* concept, with 2048 examples consisting of 11 bits, only 5 of which are relevant (i.e. they decide the value of the final 12th, or 'class' bit). Seshu (1989) states that the parity problem causes many problems and in general, accuracy and complexity suffer significantly. Seshu refers to this as the *unsplittability problem*, where no split can improve the class distribution 'vectors'. Seshu contends that the parity problem is not a rare, "academic" exercise, but one which surfaces in many guises, which he names "quasi-classical parity situations". Specifically, such a set of problems consists of situations where the unsplittability problem occurs, i.e. where monothetic attribute

selection offers no 'gain' in attribute selection.

Irrelevant bits are added to confuse an algorithm. This technique is useful for determining an algorithm's ability to ignore useless attributes and to use only those which define the relationship between class and attribute(s). Note that there is no noise (although we add some later), and no missing values. One might surmise that the parity concept, with added noise, irrelevant attributes and perhaps missing values, might be the most difficult synthetic set with which an algorithm can be expected to cope (see also Shen 1992).

### 5.2.12 Post-operative Care

This set classifies patients into 3 groups for post-operative care, viz. *intensive-care, hospital ward*, or *home*. There are 90 examples, of 8 attributes (plus class).

### 5.2.13 Soybean

This set depicts Michalski et al.'s classic induction problem (1980), but using a simplified version (no missing values etc.), derived from the larger, original data set (see Murphy et al. 1994a). There are 47 examples with 35 nominal attributes and 4 classes, depicting diseases in soy plants.

### 5.2.14 The Monk's Problems

These sets were used by Thrun et al. (1991) to compare many learning algorithms. There are 3 problems (one with noise) and each is divided into a training and a testing set (i.e. there are no random variations in this instance). There are 432 examples, 7 attributes, 2 classes $(0, 1)$ and no missing values.

## 5.3 Results

In the following, the column headings signify *cpu time* (seconds) to build the tree (T), the number of *nodes and leaves* (NL), the number of *entropy calculations* (EC)[5] , the number of *restructures* (R), the number of *aborted restructures* (AR), the number of *nodes restructured in total* (NR), and the *accuracy* of the tree (AC) in percent. In the main we refer to EC, R, AR and NR as our efficiency indicators (inductive *costs* or *performance criteria*). Full results tables are shown in Appendix B.

### 5.3.1 ID3 Versus ID4 and ID5

Below we show the results for the various 'flavours' of ID3, ID4 and ID5 for three domains, *balloons, chess* and *lung cancer*. The first simple domain illustrates their relative characteristics in a more easily analysed situation. Secondly, we show the results

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| ID3 | 0.03 | 4.8 | 6.7 | 0 | 0 | 0 | 93.3 |
| INC ID3 | 0.18 | 4.8 | 60.1 | 14 | 0 | ? | 93.3 |
| ID4 | 0.05 | 3.6 | 52.9 | 1.4 | 9 | 3.8 | 83.3 |
| $\widehat{\text{ID4}}$ | 0.04 | 4.8 | 15.2 | 0.6 | 0.8 | 0.9 | 93.3 |
| ID5 | 0.1 | 4.8 | 54.2 | 1.7 | 9.8 | 4.5 | 93.3 |
| $\widehat{\text{ID5}}$ | 0.06 | 4.8 | 16.0 | 0.7 | 0.8 | 1 | 93.3 |
| ID5R | 0.08 | 4.8 | 64.2 | 1.7 | 12.8 | 4.5 | 93.3 |
| $\widehat{\text{ID5R}}$ | 0.08 | 4.8 | 16.0 | 0.7 | 0.8 | 1.0 | 93.3 |

Table 5.1: Results for Balloons Set 2

for a chess domain (KRKPA7) to illustrate the discrepancies between the algorithms' performance criteria as one scales up to a larger, more complex domain. Finally we show relative performances for a more 'real-life' domain in which unknown noise levels, inadequate attributes and imprecise relationships may well play a part (lung cancer). Further

[5] Van de Velde and Utgoff use these as a basis for determining computational complexity

results are shown in Appendix B. One should recall that *INC_ID3* is an incremental (brute-force) ID3, whilst 'hat' versions update only on misclassifications, and ID5R recursively updates all tests in a restructured subtree. ID3 is a 'one-shot' algorithm here and makes no use of windows.

In table 5.1, one can see that ID4's accuracy has suffered due to a loss of information when discarding subtrees, not recovered by the end of the current example sets (note the smaller tree size). As expected ID3 costs least with regard to our performance criteria but obviously, is not incremental. The most expensive, with respect to time and restructures, is INC_ID3, as expected. Somewhat surprisingly, ID5R is most expensive with regard to entropy calculations. One would surmise this is due to the relatively high number of *aborts* (AR), caused by recursive sub-tree checks. The most tree restructuring is observed in INC_ID3, ID5R and then ID5. In table 5.2, one can see the marked

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| ID3 | 133.3 | 90.0 | 1278.7 | 0 | 0 | 0 | 98.1 |
| ID4 | 811.6 | 79.4 | 214635.0 | 70.3 | 5715.3 | 823.3 | 96.8 |
| $\widehat{\text{ID4}}$ | 542.0 | 66.4 | 15225.5 | 27.3 | 212.6 | 345.6 | 97.7 |
| ID5 | 2145.0 | 92.0 | 208063.0 | 122.1 | 5754.4 | 1609.1 | 98.1 |
| $\widehat{\text{ID5}}$ | 1149.4 | 87.0 | 14525.6 | 54.2 | 261.5 | 770.7 | 98.0 |
| ID5R | 2730.8 | 90.0 | 236318.0 | 178.4 | 6592.9 | 2053.1 | 98.1 |
| $\widehat{\text{ID5R}}$ | 2115.3 | 87.4 | 24957.1 | 85.8 | 555.1 | 956.5 | 98.1 |

Table 5.2: Results for KRKPA7 Chess Set

increases in run-times, entropy calculations, and so on, for the incremental algorithms over ID3. One should also note the considerable number of aborted restructures, entropy calculations and run-times for ID4, ID5 and ID5R. $\widehat{\text{ID4}}$, $\widehat{\text{ID5}}$ and $\widehat{\text{ID5R}}$ demonstrate how it is possible to reduce computation significantly whilst maintaining accuracy and even (slightly) reducing complexity. The differences between EC, R and AR totals for hat and non-hat versions of both ID4 and ID5 suggest that there are relatively few misclassifications and so the hat versions can make considerable savings. The 'R' counts

for all hat and non-hat 'pairs' are significantly different (99.9%). INC_ID3 could not

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| ID3 | 1.3 | 12.9 | 226.6 | 0 | 0 | 0 | 55.0 |
| INC ID3 | 11.3 | 12.9 | 2499.5 | 19 | 0 | ? | 55.0 |
| ID4 | 3.1 | 12.3 | 1737.1 | 5.9 | 10.7 | 22.8 | 55.0 |
| $\widehat{\text{ID4}}$ | 2.8 | 10.8 | 1263.1 | 5.9 | 2.8 | 21.5 | 55.0 |
| ID5 | 4.5 | 12.3 | 2454.4 | 9.0 | 15.2 | 37.6 | 55.0 |
| $\widehat{\text{ID5}}$ | 4.1 | 12.4 | 1806.3 | 8.5 | 4.2 | 34.2 | 56.3 |
| ID5R | 5.4 | 12.3 | 2638.5 | 9.4 | 17.7 | 38.5 | 55.0 |
| $\widehat{\text{ID5R}}$ | 5.7 | 12.4 | 1914.4 | 8.7 | 5.7 | 34.6 | 56.3 |

Table 5.3: Results for Lung Cancer Set

cope with the size of this set. It is also interesting to note the fluctuations in tree size, whilst accuracy remains relatively stable (ID4 is smallest again).

In table 5.3 one can get some feeling for the inherently under-determined nature of many real-life domains, where insufficient detail and/or data eclipses true underlying relationships. Typically, such a domain is not well understood, hence the much reduced accuracy. The trend is as before, with ID5R most expensive for entropy (EC) and aborts (AR), with INC_ID3 most expensive for time (T) and restructures (R), followed by ID5[6] . Somewhat surprisingly, $\widehat{\text{ID4}}$ is the next 'cheapest' after ID3. See also tables B.27, B.45, B.37 to B.40, and B.56 to B.58.

### Conclusion

We conclude the following points, subject to the caveats regarding inductive assertions noted at the end of §5.1.

Incremental processing is a computationally intensive process and thus algorithmic efficiency is of paramount importance. It is often not necessary to stipulate ID3-equivalence,

[6] Note t-test results for restructure totals indicate no significant difference here between hat and non-hat versions.

thereby saving effort. This is witnessed by very similar complexities and accuracies for say, $\widehat{\text{ID5}}$ and ID5R, whilst computational criteria often differ significantly.

An incremental ID3 is usually the most expensive of all algorithms, although some 'cost' facets are exceeded by ID5R (e.g. entropy calculations). ID4 frequently suffers from 'lost' information but can also be cheaper than ID5 in terms of entropy calculations and the number and extent of restructures. In general, ID5 and ID5R perform most restructures and ID5R is usually most expensive for *aborts*. Overall, the 'hat' versions save time and effort and may even find smaller, more accurate trees. Therefore, in terms of performance, we conclude that $\widehat{\text{ID5}}$ offers the best incremental algorithm, but still falls some way short of ID3. $\widehat{\text{ID5R}}$ does not appear to be of much benefit, i.e. if one requires ID3-equivalence, one should use ID5R, and if one requires a more efficient run, $\widehat{\text{ID5}}$ is better.

### 5.3.2 The JITTER family

ID5_DELAY, PSEUDO_JITTER, and their 'hat' versions are shown with *percentage* thresholds (the value of which is shown after the program name, see §4.2.2 and §4.2.3) and are abbreviated to *DL* and *PJ* respectively. Thus $\widehat{\text{DL}}$ 50 indicates ID5_$\widehat{\text{DELAY}}$ with a threshold of 50%. Percentages allow a more convenient method of determining thresholds, but some results for absolute thresholds are shown in Appendix B for completeness. It is noted that absolute thresholds allow a finer-grained choice for sets with more than 100 examples, i.e. 1% may correspond to more than one example, but in practice the use of percentages was not found to be problematic. *QUASI_JITTER* (QJ) is shown with absolute thresholds of $x$ signifying ratios of $x : 1$, correct : incorrect (also after the name). The threshold values shown for the above three algorithms have been empirically chosen to illustrate their performance characteristics.

We also show three main versions of JITTER. The first version (*J1*), is as described

initially in §4.2.5, i.e. multiple restructures are allowed, checking bottom-up, and weights are attached to branches and are compared to zero. Version 2 (*J2*) is as above plus the additions for coping with noise, i.e. *subtree weighting* and *cf biasing*. The final version (*JTD*) is as J1 except *one* restructure is permitted per example, and is determined *top-down*.

Different test sets may be used where a point is considered noteworthy, although these are still chosen to be representative (for others, see Appendix B).

A *more* efficient implementation of JITTER would easily result should a *particular* method be decided upon. That is to say, JITTER at present suffers from overheads associated with multiple testing versions[7] , and a simpler, slimmed down version will run quicker still.

### ID5_DELAY

In tables 5.4 and 5.5, all performance criteria reach a plateau at around 60% and do not change further. One can see the general decrease in time, entropy calculations and restructures etc., but also the substantial increase in tree size and decrease in accuracy. The t-test confidence levels between thresholds of 0% and 60% are: NL (99.9%), EC (99.9%), R (99.9%), AR (99.9%) and AC (99%). These indicate that thresholds cause significant differences in inductive parameters in this domain. The hat versions allow some savings to be made. That is, t-test confidence suggests there is no significant difference in size (NL) or accuracy (AC), but 95% in R, and 99.9% in AR and EC (for hat versus non-hat, for a threshold of 10%). See also tables B.13, B.14, B.15 and B.16.

In table 5.6, ID5_DELAY exhibits non-monotonic behaviour, i.e. does not steadily increase or decrease for the performance criteria, but instead shows a number of peaks and troughs. Initially one should note the substantial run-time (etc.) reduction between

[7] e.g. function pointers and multiple, short modules were used to avoid frequent recompilation.

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| DL 0 | 2.9 | 9.3 | 1510.6 | 2.0 | 28.3 | 12.5 | 86.7 |
| DL 10 | 2.6 | 9.7 | 1465.2 | 1.9 | 27.3 | 12.0 | 84.0 |
| DL 20 | 2.3 | 12.6 | 1332.5 | 1.2 | 21.0 | 9.1 | 84.7 |
| DL 30 | 2.0 | 21.5 | 1173.8 | 0.3 | 15.5 | 2.3 | 70.7 |
| DL 40 | 2.1 | 22.9 | 1156.3 | 0.2 | 13.5 | 3.3 | 69.3 |
| DL 50 | 2.0 | 25.2 | 1145.2 | 0.1 | 13.4 | 0.4 | 66.7 |
| DL 60 | 1.9 | 24.5 | 1122.0 | 0 | 13.2 | 0 | 66.7 |

Table 5.4: Results for Soybean Set

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| $\widehat{\text{DL}}$ 0 | 2.2 | 9.3 | 685.2 | 1.5 | 6.2 | 8.6 | 86.7 |
| $\widehat{\text{DL}}$ 10 | 2.1 | 10.2 | 678.0 | 1.3 | 6.2 | 6.9 | 88.0 |
| $\widehat{\text{DL}}$ 20 | 1.9 | 17.5 | 678.7 | 0.6 | 4.6 | 4.0 | 73.3 |
| $\widehat{\text{DL}}$ 30 | 2.0 | 21.5 | 686.9 | 0.3 | 3.2 | 2.5 | 70.7 |
| $\widehat{\text{DL}}$ 40 | 2.1 | 22.9 | 716.0 | 0.2 | 2.5 | 3.3 | 69.3 |
| $\widehat{\text{DL}}$ 50 | 2.0 | 25.2 | 705.3 | 0.1 | 2.4 | 0 | 66.7 |
| $\widehat{\text{DL}}$ 60 | 1.9 | 24.5 | 679.4 | 0 | 2.2 | 0 | 66.7 |

Table 5.5: Results for Soybean Set

thresholds at 0% and 5% (0% should give an indication of ID5's behaviour) although there is actually an *increase* in the number of nodes restructured (NR). The considerably reduced run-time but increased NR count indicates more nodes are being manipulated per restructure (NR/R gives a crude measure of this). This could imply that the size of restructures is not as significant to efficiency as is the act of actually restructuring. T, AR and EC counts are reduced (t-test: 99.9%) between thresholds of 0 and 10% (although NL, AC and R are not), indicating a small threshold is of benefit.

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| DL 0 | 120.0 | 158.9 | 17050.1 | 284.6 | 3274.8 | 12449.1 | 72.4 |
| DL 5 | 56.6 | 162.2 | 13608.9 | 233.4 | 2705.6 | 17758.9 | 72.6 |
| DL 10 | 44.3 | 162.0 | 11737.1 | 171.2 | 2404.6 | 13859.8 | 72.7 |
| DL 15 | 39.6 | 162.4 | 10692.4 | 151.1 | 2263.4 | 14077.9 | 72.6 |
| DL 20 | 33.9 | 161.6 | 9595.2 | 119.1 | 2076.5 | 11392.2 | 72.3 |
| DL 25 | 32.0 | 163.0 | 8786.2 | 58.7 | 1959.5 | 4783.5 | 72.4 |
| DL 30 | 29.2 | 162.0 | 7673.7 | 34.1 | 1776.5 | 3199.5 | 72.4 |
| DL 35 | 26.3 | 162.5 | 7356.7 | 21.1 | 1730.3 | 1740.5 | 72.5 |
| DL 37 | 26.1 | 162.1 | 7238.0 | 17.3 | 1710.6 | 1321.0 | 72.5 |
| DL 40 | 26.2 | 162.1 | 7238.8 | 10.7 | 1719.5 | 771.7 | 72.4 |
| DL 43 | 28.2 | 162.1 | 7029.2 | 36.1 | 1652.2 | 2393.6 | 72.7 |
| DL 45 | 26.5 | 162.1 | 6330.2 | 35.3 | 1528.4 | 2776.7 | 72.7 |
| DL 47 | 27.8 | 163.3 | 5755.5 | 25.5 | 1447.8 | 1621.6 | 72.7 |
| DL 50 | 24.2 | 162.9 | 4700.3 | 17.6 | 1268.8 | 988.4 | 72.7 |
| DL 53 | 17.5 | 162.6 | 4515.6 | 2.3 | 1221.1 | 61.6 | 72.5 |
| DL 55 | 27.1 | 162.6 | 4280.4 | 2.3 | 1188.4 | 60.6 | 72.6 |
| DL 60 | 26.0 | 162.1 | 3653.2 | 1.9 | 1087.0 | 45.6 | 72.6 |
| DL 65 | 26.9 | 162.9 | 3930.3 | 1.6 | 1158.0 | 43.0 | 72.5 |
| DL 70 | 24.2 | 163.3 | 3553.3 | 1.3 | 1094.1 | 35.9 | 72.3 |
| DL 80 | 16.4 | 162.2 | 2895.6 | 0.3 | 1035.9 | 13.2 | 72.5 |
| DL 90 | 16.0 | 163.0 | 2904.1 | 0 | 1055.9 | 0 | 72.5 |

Table 5.6: Results for 7-bit LED Set (10% noise)

Costs then generally decrease steadily (from a threshold of 5%) to a (local) minimum at about 40%, before they increase considerably (excepting entropy counts), then decrease further. This illustrates the difficulty associated with choosing a sensible threshold. See also tables B.1 to B.4. The steady reduction in entropy calculations throughout (despite

fluctuations in R etc.) suggests that they are to a certain extent, less dependent on the number of restructures than we initially thought. In fact EC counts follow the trend in AR counts, i.e. a general reduction as thresholds increase but with small 'blips' at 37-40%, 60-65% and 80-90%, suggesting that EC counts are more influenced by aborted restructure totals.

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| $\widehat{\text{DL}}$ 0 | 67.8 | 154.1 | 5630.8 | 91.6 | 904.5 | 2176.4 | 72.6 |
| $\widehat{\text{DL}}$ 10 | 39.6 | 157.2 | 4342.1 | 41.3 | 745.5 | 2643.6 | 72.6 |
| $\widehat{\text{DL}}$ 50 | 22.4 | 159.6 | 1833.1 | 7.7 | 387.8 | 368.1 | 72.6 |
| $\widehat{\text{DL}}$ 90 | 14.7 | 160.4 | 1292.3 | 0 | 339.7 | 0 | 72.5 |

Table 5.7: Results for 7-bit LED Set - 10% noise

We show the same set for ID5_$\widehat{\text{DELAY}}$ in table 5.7 to illustrate the considerable reductions for the hat version. For thresholds of 10%, comparisons of hat and non-hat versions show significant differences for R and T (95%), EC and AR (99.9%), and no difference for NL and AC, as desired. See also tables B.25, B.30, B.33, B.59.

### Conclusion

Again one must conclude that the hat version is the most economical. However, the choice of a suitable threshold is difficult and differs across domains. It may be that a low threshold of say, 5-10 % used across the board, may offer substantial savings with respect to efficiency and not decrease accuracy 'too much'. It would appear that the size of restructures is not so costly as the act of restructuring. As tree size increases and accuracy decreases, it would appear that the wrong restructures are being allowed (recall in Chapter 4 we concluded that not all $\widehat{\text{ID5}}$'s restructures are necessary). 'EC' totals do not appear to be greatly affected by R or even NL. That is, whilst they generally decrease as thresholds increase, they appear to be to a certain extent independent of fluctuations

in NL and R. EC totals follow AR totals more closely, as one might expect, aborts consist of entropy calculations only.

It would also appear that there is a trade-off between efficiency and size/accuracy, i.e. as R, EC, AR and NR decrease, NL often increases and AC decreases.

**PSEUDO_JITTER**

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| PJ 0 | 2.9 | 7.7 | 1450.8 | 3.0 | 33.0 | 9.9 | 93.3 |
| PJ 10 | 2.8 | 9.8 | 1547.2 | 1.6 | 24.9 | 12.2 | 86.0 |
| PJ 20 | 2.6 | 19.2 | 1737.7 | 0.5 | 16.4 | 5.2 | 73.3 |
| PJ 30 | 2.7 | 22.9 | 1858.6 | 0.2 | 13.7 | 2.7 | 69.3 |
| PJ 40 | 2.6 | 24.5 | 1865.5 | 0 | 13.2 | 0 | 66.7 |
| $\widehat{\text{PJ}}$ 0 | 1.9 | 7.9 | 500.1 | 2.3 | 7.4 | 6.3 | 94.0 |
| $\widehat{\text{PJ}}$ 10 | 2.0 | 21.2 | 779.4 | 0.3 | 3.3 | 1.7 | 69.3 |
| $\widehat{\text{PJ}}$ 20 | 2.1 | 22.9 | 853.9 | 0.2 | 2.7 | 1.9 | 69.3 |
| $\widehat{\text{PJ}}$ 30 | 2.1 | 24.5 | 861.0 | 0 | 2.2 | 0 | 66.7 |

Table 5.8: Results for Soybean Set

In table 5.8, PSEUDO_JITTER reaches a plateau earlier at around 30-40%. As above, T, R and AR reduce as the threshold increases, as one would expect. As with ID5_DELAY, size increases and accuracy decreases, indicating that the restructures chosen are not ideal, and confirming our intuition regarding the efficiency trade-off. Comparing cost parameters for thresholds of 0% and 10% gives at least 99% certainty of significant difference for R, NL, EC, and AR, but not T or AC. Thus thresholds can increase efficiency. The hat version aids efficiency but also appears to affect efficiency parameters more (e.g. compare 10% thresholds, hat and non-hat versions for NL counts). That is, performance characteristics reach a plateau earlier at 30% as opposed to 40%. In general we take a restructure score of zero to indicate a 'plateau' has been reached. This has been empirically found to indicate that no further improvements will be forthcoming.

Comparing hat and non-hat versions for a 10% threshold indicates significant differ-

ence of at least 99% confidence for all parameters (even though NL and AC are worse for PSEUDO_JITTER). One should note however, that entropy calculations generally *increase* with the threshold. This is caused by the method for determining whether a restructure is necessary, which *necessitates* the recalculation of entropy (§4.2.3). See also tables B.17 to B.20.

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| PJ 0 | 306.6 | 159.1 | 16717.7 | 217.1 | 3294.0 | 3330.9 | 72.6 |
| PJ 10 | 47.0 | 162.8 | 17902.3 | 15.5 | 2663.6 | 450.8 | 72.6 |
| PJ 50 | 31.9 | 162.3 | 18091.2 | 1.8 | 1810.5 | 69.9 | 72.6 |
| PJ 90 | 28.3 | 163.0 | 18157.5 | 0 | 1694.8 | 0 | 72.5 |
| $\widehat{\text{PJ}}$ 0 | 100.3 | 152.8 | 5411.2 | 127.8 | 890.9 | 1841.6 | 72.5 |
| $\widehat{\text{PJ}}$ 10 | 25.8 | 158.0 | 6046.4 | 5.9 | 716.0 | 149.0 | 72.3 |
| $\widehat{\text{PJ}}$ 50 | 18.1 | 159.7 | 6206.4 | 0.4 | 522.1 | 7.1 | 72.5 |
| $\widehat{\text{PJ}}$ 90 | 18.5 | 160.4 | 6194.9 | 0 | 521.0 | 0 | 72.5 |

Table 5.9: Results for 7-bit LED Set - 10% noise

In table 5.9 there is again a marked difference between the unrestricted version (threshold = 0%) and low thresholds, indicating the potential for increased efficiency from waiting a relatively small number of restructure requests. Significant differences occur between T, R, EC, and AR (99.9%), between thresholds of 0% and 10% (but not NL and AC). Once again the hat version offers significant savings (comparing with the non-hat version for a 10% threshold: T, R, EC and AR at 99.9% confidence, NL at 95%, none for AC). There are considerable decreases in the *proportion* of restructures to aborts here, e.g. decreasing to approximately $\frac{1}{1000}$ for PJ(50). This may indicate that PSEUSO_JITTER is not stopping many unnecessary changes (if any), or even that its modus operandi actually increases them. Ideally we wish R, AR and NR to decrease 'in line'. Again EC and AR seem most tightly coupled. See also tables B.5 to B.8, B.60, B.25, B.30 and B.34.

### Conclusion

In conclusion, the number of entropy calculations rises as the threshold increases for this type of counter quantity. Thresholds have a significant effect, but the choice of a suitable threshold value is difficult and differs across domains. The threshold here is more stringent in that a greater effect is noticed for the same percentage change in comparison with ID5_DELAY (also a 'plateau' is reached earlier). This is to be expected as the counter deals with restructures, not examples, but indicates that care is required. Indeed, using a hat version appears to compound the effect and increase the magnitude of change, but generally saves expense. There have been large decreases in the proportion of R : AR, (i.e. AR counts increase) which is not ideal. Aborts are wasted effort even if relatively inexpensive as compared to restructures and again appear to cause rising EC totals. One would surmise the wrong restructures are still being allowed and/or the correct ones disallowed. Overall then, it would appear that this counter quantity is not as effective as that of ID5_DELAY. As before, rising complexity and decreasing accuracy may occur with rising thresholds, but a low threshold setting may be beneficial in most domains, combined with the hat version.

## QUASI_JITTER

Next we compare QUASI_JITTER, with and without the *accuracy criterion reduction* ("-TDEC" indicates *without*), where our accuracy criterion is successively reduced as one climbs a tree path. As expected, QUASI_JITTER(1) accepts *most* node purities and so accuracy suffers (table 5.10). As the threshold increases, so do the costs of the algorithm, as it tries to maintain the higher degree of node purity. Comparing parameters between thresholds of '1' and '4', significant differences occur between T (99% confidence), R, NL, AR and AC (99.9%) and EC (95%). Note how tree size decreases

and accuracy increases (vis-à-vis the previous algorithms), as the threshold *increases*.

QUASI_JITTER without the accuracy criterion reduction looks to be generally more

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| QJ 1 | 1.5 | 22.5 | 318.4 | 0.2 | 0 | 2.5 | 68.0 |
| QJ 1.25 | 1.5 | 11.5 | 289.9 | 2.1 | 0.2 | 3.8 | 90.7 |
| QJ 1.50 | 1.5 | 11.5 | 289.9 | 2.1 | 0.2 | 3.8 | 90.7 |
| QJ 1.75 | 1.7 | 9.6 | 317.2 | 2.4 | 0.2 | 7.1 | 94.7 |
| QJ 2 | 1.7 | 8.7 | 331.4 | 2.6 | 0.2 | 9.0 | 92.7 |
| QJ 4 | 2.0 | 7.7 | 411.4 | 3.5 | 1.6 | 10.6 | 93.3 |
| QJ-TDEC 1 | 1.8 | 10.0 | 337.5 | 1.1 | 0 | 10.1 | 87.3 |
| QJ-TDEC 1.25 | 1.7 | 8.0 | 322.8 | 2.7 | 0.1 | 9.1 | 91.3 |
| QJ-TDEC 1.50 | 1.8 | 8.0 | 353.9 | 3.0 | 0.2 | 10.0 | 91.3 |
| QJ-TDEC 1.75 | 1.9 | 8.0 | 386.1 | 3.2 | 1.0 | 10.5 | 91.3 |
| QJ-TDEC 2 | 1.9 | 7.7 | 400.0 | 3.3 | 1.1 | 11.3 | 93.3 |
| QJ-TDEC 4 | 2.3 | 7.7 | 603.6 | 3.9 | 6.0 | 12.4 | 93.3 |

Table 5.10: Results for Soybean Set

expensive but more accurate in a number of places, especially for low thresholds (although the only *statistically* significant difference for a threshold of '1.25', is in NL totals: 95%). Using the same accuracy criterion setting (node purity), no matter what the level of the

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| $\widehat{\text{QJ}}$ 1 | 1.0 | 24.3 | 308.0 | 0.1 | 0 | 0.4 | 66.7 |
| $\widehat{\text{QJ}}$ 1.25 | 1.1 | 10.1 | 314.2 | 2.0 | 0 | 5.6 | 91.3 |
| $\widehat{\text{QJ}}$ 1.5 | 1.1 | 10.1 | 307.5 | 2.0 | 0 | 6.2 | 91.3 |
| $\widehat{\text{QJ}}$ 1.75 | 1.1 | 9.4 | 317.6 | 2.1 | 0 | 7.3 | 94.7 |
| $\widehat{\text{QJ}}$ 2 | 1.1 | 8.7 | 308.0 | 2.0 | 0 | 7.9 | 92.7 |
| $\widehat{\text{QJ}}$ 4 | 1.0 | 8.0 | 313.9 | 2.5 | 0.8 | 8.6 | 94.0 |

Table 5.11: Results for Soybean Set

tree is counter intuitive. We would surmise that in this case, the comparatively higher level of desired purity further up the tree is causing more restructures, nearer the top, hence the larger 'R', 'AR' and 'NR' totals. In fact a quick check of *NR/R* (i.e. a crude measure of the number of nodes per restructure) confirms more per restructure for lower

thresholds, for QJ-TDEC generally. Again, this is counter to our aims. Table 5.11 shows savings for $\widehat{QJ}$ over QJ. For a threshold of '1', the only significant difference is run-time (99.9% confidence), whereas for a threshold of '4', R, EC and AR are also 95% certain to be different (NL and AC are not). See also tables B.21 to B.23.

In table 5.12, once again costs rise with the threshold. The number of entropy calculations increases substantially as do aborts, whilst actual restructures rise more sedately. Removing the threshold reduction mechanism again results in higher costs over 'QJ' and '$\widehat{QJ}$'. For a threshold of '2', R, EC and AR differ with confidence 99.9%, and T with 95% (NL and AC do not), between $\widehat{QJ}$ and QJ. See also tables B.9 to B.11 and B.26, B.29, B.31, B.35, B.46, B.48, B.50, and B.61.

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| QJ 1 | 21.0 | 158.6 | 1949.1 | 35.6 | 148.7 | 420.7 | 72.6 |
| QJ 1.5 | 58.9 | 156.6 | 7134.3 | 79.3 | 702.8 | 1392.1 | 72.7 |
| QJ 2 | 88.8 | 157.0 | 12828.5 | 107.3 | 1449.3 | 2071.7 | 72.6 |
| QJ-TDEC 1 | 97.6 | 156.7 | 14216.0 | 80.3 | 1552.4 | 2042.3 | 72.7 |
| QJ-TDEC 1.5 | 104.5 | 156.5 | 16217.2 | 112.7 | 1886.1 | 2295.8 | 72.7 |
| QJ-TDEC 2 | 109.4 | 156.4 | 17304.9 | 126.5 | 2111.5 | 2403.6 | 72.7 |
| $\widehat{QJ}$ 1 | 20.8 | 159.7 | 1295.5 | 21.7 | 73.6 | 287.8 | 72.6 |
| $\widehat{QJ}$ 1.50 | 36.1 | 158.4 | 2978.6 | 43.0 | 229.3 | 662.9 | 72.7 |
| $\widehat{QJ}$ 2 | 57.6 | 157.7 | 4869.9 | 59.5 | 403.7 | 1147.9 | 72.7 |

Table 5.12: Results for 7-bit LED Set - 10% noise

### Conclusion

QUASI_JITTER potentially allows many more restructures. For a threshold of 2:1, which means that 67% of a node's examples must be in the majority class, it must work hard to avoid restructuring. One can see the different emphasis between this and previous algorithms. For QUASI_JITTER we assume that we will allow restructuring until it is not necessary whereas for ID5_$\widehat{DELAY}$ and PSEUD$\widehat{O}$_JITTER we stipulate that restructures

will not be done until absolutely necessary. QUASI_JITTER with a threshold of '1.0' performed well in most domains reported here and occasionally better than JITTER. However, such a threshold value is analogous to one of 100% for PSEUDO_JITTER or ID5_DELAY in that no restructures are allowed at all (i.e. a ratio of 0:1 or 1:1 will simply accept any 'accuracy' at a node for two classes, and most for more than two classes). It is for this reason that we discount this particular threshold value. Generally, costs increase as the threshold increases indicating higher levels of node purity are being achieved. One must be careful not to set any threshold 'too high' otherwise QUASI_JITTER becomes very expensive. QUASI_JITTER(2) appears to be the upper range at which QUASI_JITTER becomes uneconomical. Whilst its run-times are still good compared to $\widehat{\text{ID5}}$, the number of entropy calculations, restructures and aborted restructures start to exceed even $\widehat{\text{ID5}}$.

Not decreasing the accuracy criterion as one climbs a tree is counter-intuitive and causes restructures at higher levels than otherwise would be the case. This version also increases costs generally, to not much benefit elsewhere, and therefore we use it no further. Again, thresholds cause significant change in cost parameters. The choice of a threshold is problematic, but a small, constant value may be beneficial, allied to the hat version as before. Individual node purity appears to be a good indicator of global accuracy (and indirectly even size) and altering it has a direct effect on accuracy and tree size.

### Comparisons

So far we have discussed ID5_DELAY, PSEUDO_JITTER and QUASI_JITTER in isolation. Here we compare selected versions and settings against JITTER (J1 and JTD). In the tables below we generally select a 'good' and 'bad' threshold for ID5_DELAY, PSEUDO_JITTER and QUASI_JITTER for comparison (although not the extremes of 0% and 100% etc.).

Table 5.13 shows results for the parity problem, arguably one of the worst domains (see also tables B.33 to B.36). The first point to note is JITTER's excellent complexity and accuracy. However, JITTER's costs are not as low as they might be, e.g. run-time. JITTER scores well on ECs (lowest of all on average), whilst ID5_DELAY and PSEUDO_JITTER do well for restructures on average. We compare J1 against DL(5),

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| DL 5 | 516.8 | 1619.9 | 22672.0 | 165.1 | 134.9 | 82141.7 | 62.1 |
| DL 50 | 96.1 | 1942.3 | 9531.3 | 2.8 | 1.1 | 452.6 | 41.6 |
| PJ 10 | 158.9 | 1631.3 | 40661.0 | 6.6 | 1247.7 | 1179.3 | 55.4 |
| PJ 30 | 73.2 | 1953.8 | 49246.0 | 0.5 | 1275.3 | 95.2 | 39.9 |
| QJ 1 | 48.1 | 2075.6 | 2815.9 | 0 | 0 | 0 | 34.4 |
| QJ 4 | 443.3 | 648.9 | 58149.5 | 553.3 | 1068.5 | 18140.3 | 83.3 |
| J1 | 313.3 | 592.8 | 8182.4 | 457.4 | 396.7 | 7795.3 | 90.5 |
| JTD | 558.1 | 624.6 | 7996.2 | 274.4 | 474.5 | 10312.3 | 86.4 |

Table 5.13: Results for Parity Set

PJ(10) and QJ(4) (chosen for maximum accuracy) using t-tests for judging the confidence in differences as follows. For DL(5), R, NL, EC, AR and AC there was 99.9% confidence that the mean values are significantly different. N.B. one should note the R and AR counts for DL(5) are therefore significantly less than J1, although overall J1 is better. Similarly for PJ(10): T, R and NR are better, but overall J1 is better. QJ(4) gave 99.9% for EC and AR, 95% for AC but no significant difference for T, R and NL.

Comparing J1 and JTD one can see that bottom-up restructure checking results in more restructures but fewer nodes restructured overall, and fewer aborts for a quicker run-time. Considering JITTER's accuracy and complexity we conclude that JITTER represents a good compromise solution to our aims of increased efficiency (for this set at least). See also Appendix B, table B.36 for a very favourable comparison with ID3 and ID5.

ID5_DELAY, PSEUDO_JITTER and QUASI_JITTER are subject to considerable

variation across thresholds, and can produce large inaccurate trees. One can see the extremes possible with $\widehat{QJ}(1)$ and $\widehat{QJ}(4)$. A large tree ($\widehat{QJ}(1)$) with very few entropy calculations, no restructures, and a maximum path length of 12 (see Appendix B, table B.35) indicates relatively few entropy calculations are required to build a whole tree. Conversely, a 'small' tree, many restructures, and thousands of entropy calculations ($\widehat{QJ}(4)$) indicates the considerable costs of restructuring. As one can see between $\widehat{QJ}(4)$ and JTD say, many restructures are unnecessary.

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| $\widehat{DL}$ 10 | 1212.9 | 138.1 | 12192.6 | 7.2 | 1.4 | 664.6 | 97.4 |
| $\widehat{DL}$ 50 | 288.0 | 210.6 | 8040.4 | 0.1 | 0 | 9.2 | 96.3 |
| $\widehat{PJ}$ 0 | 1084.7 | 87.0 | 14525.6 | 54.2 | 261.5 | 770.7 | 98.0 |
| $\widehat{PJ}$ 10 | 266.1 | 211.9 | 21415.2 | 0.0 | 113.9 | 0.0 | 96.2 |
| $\widehat{QJ}$ 1.25 | 264.0 | 179.7 | 3360.3 | 8.2 | 1.9 | 20.8 | 96.7 |
| $\widehat{QJ}$ 4.0 | 393.7 | 88.8 | 9783.5 | 27.2 | 47.7 | 274.4 | 96.6 |
| J1 | 257.8 | 190.9 | 3256.9 | 7.9 | 2.0 | 18.3 | 96.8 |
| JTD | 277.9 | 190.9 | 3227.7 | 7.1 | 2.1 | 15.9 | 96.8 |

Table 5.14: Results for KRKPA7 Chess Set

In table 5.14 accuracy varies little but tree size fluctuates wildly. Again ID5_$\widehat{DELAY}$, PSEUD$\widehat{O}$_JITTER and QUASI_$\widehat{JITTER}$ differ considerably across the threshold values shown, (in particular note the difference for PSEUD$\widehat{O}$_JITTER for a threshold change of 10%). QUASI_$\widehat{JITTER}$ and JITTER both have consistently low run-times, whilst JITTER shows consistently low EC, R, AR and NR counts. JITTER's node counts are disappointing but could be worse (e.g. $\widehat{PJ}$). Again we compare parameters for J1 versus $\widehat{DL}(10)$, $\widehat{PJ}(10)$ and $\widehat{QJ}(4)$, and find the following differences between means. For $\widehat{DL}(10)$, T, NL and EC give 99.9% confidence, AC gives 95%, whereas for R and AR there is no significant difference. Note that AC and NL are worse for J1. $\widehat{PJ}(10)$ gives 99.9% confidence for R, EC and AR, and 95% for NL (T and AC are not significantly different). Here R is worse for J1, but the rest are better. $\widehat{QJ}(4)$ gives 99.9% for T, R, NL, EC

and AR (AC is not significantly different). Note that NL is worse here. J1 and JTD are similar and so comparing bottom-up and top-down restructure checking is inconclusive in this domain. We would again tentatively suggest that JITTER is a compromise solution to our aims of efficient induction, without the need to set thresholds. (See also tables B.26, B.28, B.30, B.32, and B.61 etc.).

### Conclusion

We tentatively conclude the following points. For ID5_DELAY and PSEUDO_JITTER, percentage thresholds are more convenient than absolute, unless one requires extra-fine precision in large example sets. In a truly incremental setting where the number of examples is not known a priori, setting sensible absolute thresholds would be virtually impossible.

Generally, as thresholds increase (ID5_DELAY, PSEUDO_JITTER) so does the size of a tree, often with a concomitant decrease in accuracy. We attribute this to the stopping of *necessary*, and/or the permitting of *unnecessary* restructures, i.e. we still require more selectivity. This also illustrates an inherent trade-off between 'effort' and size/accuracy.

ID5_DELAY shows some non-monotonic behaviour which may indicate its thresholds are unreliable. PSEUDO_JITTER's thresholds appear to be more stringent as its cost values reach a plateau more quickly (as one would expect, the number of restructures will be somewhat less than the number of examples). Moreover, the hat version may compound this effect.

Generally, the selection of a threshold is not straightforward and would ideally require experimentation for each, separate example set. This seems counter-intuitive and is contrary to our aims of increased efficiency. On the other hand, a low threshold of say 5-10% can offer considerable efficiency savings across the board, although this differs from domain to domain. Considerable variation across threshold values is observed for

ID5_DELAY, PSEUDO_JITTER and QUASI_JITTER. Hence we believe an automatic, possibly dynamically variable threshold is required.

Entropy calculations increase considerably when aborted restructures increase. We thereby conclude that aborts are a significant source of wasted efficiency. PSEUDO_JITTER does not appear to reduce the number of aborts as much as it might, implying that one could choose a more informative quantity on which to base the counter. That is, PSEUDO_JITTER's counter works on the number of restructure requests, and as there is still a relatively high number of these, efficiency could be improved further. It would also appear that the 'size' of restructures has less effect on efficiency than the actual process of restructuring, and thus it may be more beneficial to allow one restructure, checked from the top-down, as in JTD. Moreover, it would appear that relatively few entropy calculations are needed to grow a tree and the largest proportion are in actual and aborted restructures.

PSEUDO_JITTER generally leads to increased entropy calculations, whilst QUASI_JITTER typically increases costs overall as greater purity is required, occasionally without the desired increased accuracy. The omission of an accuracy criterion reduction mechanism for QUASI_JITTER causes inappropriate behaviour. Overall, $\widehat{\text{QUASI\_JITTER}}$'s accuracy is at least as good as that achieved by $\widehat{\text{PSEUDO\_JITTER}}$ and $\widehat{\text{ID5\_DELAY}}$ (the hat versions are the most important). This is observed for the LED domain (tables B.4, B.8, B.11); for the soybean domain (tables B.16, B.20 and B.23); for the multiplexor domain (tables B.30 and B.31); for the parity domain (tables B.33 to B.35); but not the KRKPA7 chess domain (tables B.25 and B.26). Therefore we feel justified in saying that our accuracy criterion is in fact a plausible model for checking for necessary tree restructures. That is to say, a previous comment criticised ID5_DELAY and PSEUDO_JITTER for being only weakly based on information afforded by tree quality, whereas QUASI_JITTER makes the use of such information more explicit, and demon-

strates the benefit. Altering node purity directly affects accuracy and size and is an effective means of deciding when tree quality (i.e. accuracy) has deteriorated.

The hat versions of each algorithm allow considerable efficiency savings to be made but one cannot conclude that, as with $\widehat{\text{ID5}}$ etc., smaller more accurate trees may often result.

JITTER's NL counts can suffer and some other values are not as low as others but *overall* JITTER appears to be a good compromise solution, sometimes producing excellent results. It is not obvious whether top-down or bottom-up restructure checking is more beneficial. Overall then, we feel that JITTER, and possibly QUASI_JITTER, offer the best prospects of achieving increased efficiency over *all* cost parameters, for *all* test sets, whilst primarily maintaining accuracy.

### 5.3.3 JITTER and QUASI_JITTER Versus ID3 and $\widehat{\text{ID5}}$

Generally speaking we use the 'hat' version of ID5 (Utgoff 1989a). It can be seen (Utgoff 1989a, Conroy et al. 1994, and ff.) that $\widehat{\text{ID5}}$ reduces the number of entropy calculations, run-time, restructures, and even the number of tree nodes. Thus $\widehat{\text{ID5}}$ and ID3 results are shown as benchmarks for our algorithms. QUASI_JITTER thresholds are chosen to maximise accuracy, from the selection of thresholds used during testing, over as many test sets as possible to obviate re-runs (and thus inevitably some are compromises).

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| ID3 | 34.6 | 218.0 | 618.9 | 0.0 | 0.0 | 0.0 | 99.0 |
| $\widehat{\text{ID5}}$ | 1090.8 | 446.3 | 7344.2 | 179.9 | 411.3 | 9950.9 | 94.6 |
| $\widehat{\text{QJ}}$ 2 | 35.0 | 140.2 | 1339.8 | 24.3 | 14.0 | 135.5 | 99.6 |
| J1 | 37.5 | 313.8 | 1520.9 | 43.8 | 19.1 | 117.5 | 97.7 |
| JTD | 42.9 | 316.4 | 1442.3 | 36.1 | 17.3 | 105.8 | 97.7 |

Table 5.15: Results for Multiplexor Set

Note how run-times are all significantly reduced over $\widehat{\text{ID5}}$ (table 5.15) to be only slightly more than the non-incremental ID3. $\widehat{\text{QJ}}(2)$ even improves accuracy over ID3 (95% confidence that this is significant), and indeed is 'best' overall (e.g. there is no significant difference between T and NL counts compared to ID3). All our algorithms consistently out-perform $\widehat{\text{ID5}}$ in this table, on all counts. $\widehat{\text{QJ}}(2)$ improves over $\widehat{\text{ID5}}$ with confidence 99.9% for all of T, R, NL, EC, AR, AC. J1 achieves at least 99% for the same parameters. Again there is little to choose between J1 and JTD. See also tables B.31 and B.32.

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| ID3 | 10.1 | 238.4 | 1922.3 | 0 | 0 | 0 | 64.8 |
| $\widehat{\text{ID5}}$ | 90.8 | 149.7 | 10669.8 | 53.8 | 145.0 | 504.9 | 66.5 |
| $\widehat{\text{QJ}}$ 2 | 20.0 | 198.6 | 6763.7 | 22.5 | 19.0 | 247.2 | 65.3 |
| J1 | 64.1 | 153.4 | 9933.8 | 62.5 | 129.7 | 733.9 | 66.4 |
| JTD | 31.7 | 154.1 | 5473.9 | 22.8 | 64.6 | 232.8 | 66.1 |

Table 5.16: Results for Cardio-Vascular Set

In table 5.16 one can see how all algorithms improve complexity and accuracy over ID3. J1 saves considerable run-time (T) over $\widehat{\text{ID5}}$ (95% confidence) despite increasing the restructure count (no significant difference). In fact, there are no other significant differences for $\widehat{\text{ID5}}$. JTD on the other hand significantly reduces T over $\widehat{\text{ID5}}$ (99.9%) together with EC, R, and AR (99.9%), implying that top-down restructure checking is more beneficial for this domain. QUASI_JITTER is also very 'cheap' but NL is still quite high (T, R, NL, EC and AR are all significantly different with confidence 99.9%, whilst AC is not, compared to $\widehat{\text{ID5}}$). In table 5.17, all algorithms improve complexity but not accuracy over ID3. J1, JTD and $\widehat{\text{QJ}}$ are again close to ID3's run-time and $\widehat{\text{QJ}}$'s costs are closest. In comparing $\widehat{\text{ID5}}$ to J1, JTD and $\widehat{\text{QJ}}(2)$, T, R, EC and AR are all significantly different (confidence 99.9%). Furthermore, J1 and JTD's NL and AC totals are not significantly different from $\widehat{\text{ID5}}$, whereas $\widehat{\text{QJ}}$ 2's NL count is (99%).

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| ID3 | 3.1 | 237.4 | 289.7 | 0 | 0 | 0 | 58.5 |
| $\widehat{\text{ID5}}$ | 17.3 | 149.7 | 2586.0 | 48.1 | 155.4 | 558.7 | 56.7 |
| $\widehat{\text{QJ}}$ 2 | 3.0 | 161.6 | 658.2 | 10.3 | 10.8 | 44.1 | 57.7 |
| J1 | 5.2 | 152.4 | 1178.2 | 26.6 | 61.5 | 145.1 | 57.7 |
| JTD | 4.8 | 147.0 | 959.7 | 17.4 | 45.5 | 93.0 | 57.1 |

Table 5.17: Results for Breast Cancer Set

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| ID3 | 1.3 | 12.9 | 226.6 | 0 | 0 | 0 | 55.0 |
| $\widehat{\text{ID5}}$ | 4.1 | 12.4 | 1806.3 | 8.5 | 4.2 | 34.2 | 56.3 |
| $\widehat{\text{QJ}}$ 2 | 2.2 | 18.0 | 1038.6 | 4.2 | 0.4 | 15.1 | 31.3 |
| J1 | 2.7 | 17.6 | 1051.0 | 5.3 | 0.6 | 18.3 | 32.5 |
| JTD | 2.4 | 18.4 | 1002.6 | 4.3 | 0.5 | 13.1 | 33.8 |

Table 5.18: Results for Lung Cancer Set

Finally we show a domain in which JITTER and QUASI_JITTER do not fare so well. J1, JTD and QJ save considerable expense, but accuracy and complexity suffer significantly (table 5.18). Comparing ID5 with each of J1, JTD and QJ for T, R, NL, EC, AR and AC, one can see that *all* totals are significantly different (with confidence at least 95%). This confirms that whilst our algorithms save significant expense, the worsening size and accuracy are also significant. See also tables B.11, B.12, B.23, B.24, B.26 to B.29, B.31, B.32, B.35, B.36, and B.37 to B.62.

### Conclusion

We have shown how, on balance, ID5 can waste significant effort during induction. JITTER and QUASI_JITTER can significantly reduce inductive costs whilst on the whole maintaining accuracy. Indeed, JITTER's and QUASI_JITTER's increased efficiency *often* makes them more comparable to the non-incremental ID3 than to ID5. We would suggest that top-down restructure checks (JTD) are in fact more beneficial. Bottom-up checking (J1) often saves computation but may leave unsuitable tests further up, hence the potential for more restructures overall. JITTER is often more complex than ID5, although there are a number of notable exceptions in domains where the received wisdom is that ID3 etc. are poor classifiers. In general we would reiterate that there is a trade-off between restructure counts and node counts, which must be optimised per domain.

## 5.3.4 The Effects of Noise

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| ID3 | 10.0 | 216.0 | 306.2 | 0.0 | 0.0 | 0.0 | 47.9 |
| ID5 | 225.7 | 215.2 | 9657.6 | 253.2 | 1644.0 | 4990.2 | 47.9 |
| J1 | 259.1 | 219.9 | 15346.3 | 387.2 | 2653.4 | 6678.3 | 47.8 |
| J2 | 13.7 | 216.8 | 529.0 | 10.8 | 13.1 | 72.0 | 48.1 |

Table 5.19: Results for 7 Segment LED Set - 20% noise

Here we illustrate the effects of noise on ID3, $\widehat{ID5}$ and JITTER. We also introduce the previously mentioned version J2 which has a number of additions for coping with noise (§4.2.5).

In table 5.19, one can see where the LED set has caused JITTER's performance costs (J1) to increase beyond those of $\widehat{ID5}$. The simple additions to J2 have caused a stark change and are again, more similar to ID3. Differences in T, R, EC and AR give confidence levels of 99.9%, NL gives 99%, whilst AC is not significantly different (J1 to J2). Further evidence is offered in table 5.20. In table 5.21, $\widehat{ID5}$ could not cope and

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| ID3 | 40.7 | 835.6 | 1593.0 | 0 | 0 | 0 | 74.4 |
| $\widehat{ID5}$ | 2244.0 | 861.1 | 16045.9 | 382.9 | 1104.4 | 30215.2 | 74.0 |
| J1 | 88.4 | 884.0 | 6634.0 | 282.6 | 437.3 | 1876.6 | 74.4 |
| J2 | 52.6 | 920.8 | 3363.1 | 106.8 | 105.1 | 362.6 | 74.0 |

Table 5.20: Results for Multiplexor Set - 50% noise

ran out of memory, causing every run to terminate (for another example of an overly complex task for ID5 see Shen 1992). This illustrates JITTER's parsimonious behaviour and how more complex problem domains become tractable as a result. We attribute

| ALG | T | NL | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|
| ID3 | 37.9 | 1262.4 | 1987.3 | 0 | 0 | 0 | 62.4 |
| J1 | 2257.9 | 1015.5 | 16335.6 | 856.7 | 923.4 | 25020.7 | 76.7 |
| J2 | 100.8 | 1255.4 | 6853.3 | 384.6 | 306.7 | 2615.6 | 64.8 |
| JTD | 410.6 | 855.1 | 8415.0 | 290.9 | 493.5 | 6692.0 | 79.5 |

Table 5.21: Results for Parity Set - 10% noise

this effect to JITTER's reduced propensity for tree change and the consequently reduced use of recursive tree restructures. JTD shows how top-down restructure checking can still 'cope'. A similar effect was noted in other domains and it was often necessary to run $\widehat{ID5}$ on a 'larger' Sun workstation to avoid disk thrashing problems. This does not

generally cause even higher run-times as these are collated from *cpu time* which does not take account of the time a program is 'swapped out to disk' (virtual memory).

In conclusion, we suggest that more tests are required but our feelings are that JITTER is admirably frugal with resources, especially time and memory. J1's resilience to noise is perhaps suspect, i.e. its expense for the LED domain and the reduced margins for the parity and multiplexor domains with added noise suggest that $\widehat{ID5}$ may be more robust. However, the addition of some simple heuristic measures greatly improves this state of affairs (i.e. J2).

### 5.3.5 The Effects of Pruning

| PERIOD | ALG | NL | L | AC |
|---|---|---|---|---|
| BEFORE | $\widehat{ID5}$ | 87.0 | 45.0 | 98.0 |
| AFTER | $\widehat{ID5}$ | 64.5 | 33.7 | 98.5 |
| BEFORE | J1 | 190.9 | 97.6 | 96.8 |
| AFTER | J1 | 121.4 | 62.5 | 97.6 |

Table 5.22: Results for KRKPA7 Chess Set with Pruning

Previously, we show how JITTER's complexity can suffer with respect to ID3 and $\widehat{ID5}$. Here we show salient figures for some test sets after pruning to illustrate the difference in fit for some of the algorithms. The method we use is Quinlan's (1987b) *reduced error pruning*. It is an inherently simple method which is quicker than say, cross-validation (Breiman et al. 1984) and therefore more suitable for our current purposes. The

| PERIOD | ALG | NL | L | AC |
|---|---|---|---|---|
| BEFORE | $\widehat{ID5}$ | 446.3 | 223.4 | 94.6 |
| AFTER | $\widehat{ID5}$ | 320.6 | 160.8 | 96.0 |
| BEFORE | J1 | 313.8 | 157.4 | 97.7 |
| AFTER | J1 | 262.2 | 131.8 | 98.3 |

Table 5.23: Results for Multiplexor Set with Pruning

proportions of nodes/leaves for $\widehat{\text{ID5}}$ and JITTER *before and after* pruning give conflicting changes. That is, JITTER's trees shed relatively more nodes than $\widehat{\text{ID5}}$ during pruning in some cases, whilst not in others. In general it would appear that node and leaf counts are still *proportionately* similar. We had pondered whether JITTER's occasionally larger trees might have relatively more nodes pruned away to converge (ultimately) on $\widehat{\text{ID5}}$'s complexity values. This does not appear to be so. This would imply that JITTER's trees are not just simply more 'leafy' than $\widehat{\text{ID5}}$'s, with a number of extra attribute tests on the 'end' of each path, but are in fact quite different. That is, JITTER's sequences of test attributes are required to maintain accuracy whilst pruning, otherwise trees would be more similar once pruned. However, a better pruning technique may show this to be wrong.

### 5.3.6 Analysis of Intermediate Quantities

Here we give a more graphical illustration of JITTER's costs versus those of $\widehat{\text{ID5}}$, for the lymphography test set. We plot four graphs which show *intermediate* accuracy (fig. 5.1), complexity (fig. 5.2), restructures (fig. 5.3) and entropy calculations (fig. 5.4). Recall that *intermediate* quantities are calculated after the addition of each example. In fig. 5.1, one can see how JITTER (the J1 version) reduces the frequency and size of changes for $\widehat{\text{ID5}}$ (i.e. decreasing volatility and instability). Such a "learning curve" (Kibler et al. 1988) shows accuracy as a function of the number of examples so far. One can use these to predict the asymptotic accuracy and number of examples required to realise the asymptote. Such a prediction is harder for ID5. It is conceivable that the rate of change of learning accuracy could allow an algorithm to decide when to stop learning, i.e. if it is still necessary (diminishing returns to proportions). Again this is harder with ID5. As previously noted, JITTER's node counts often exceed those of $\widehat{\text{ID5}}$ (fig.

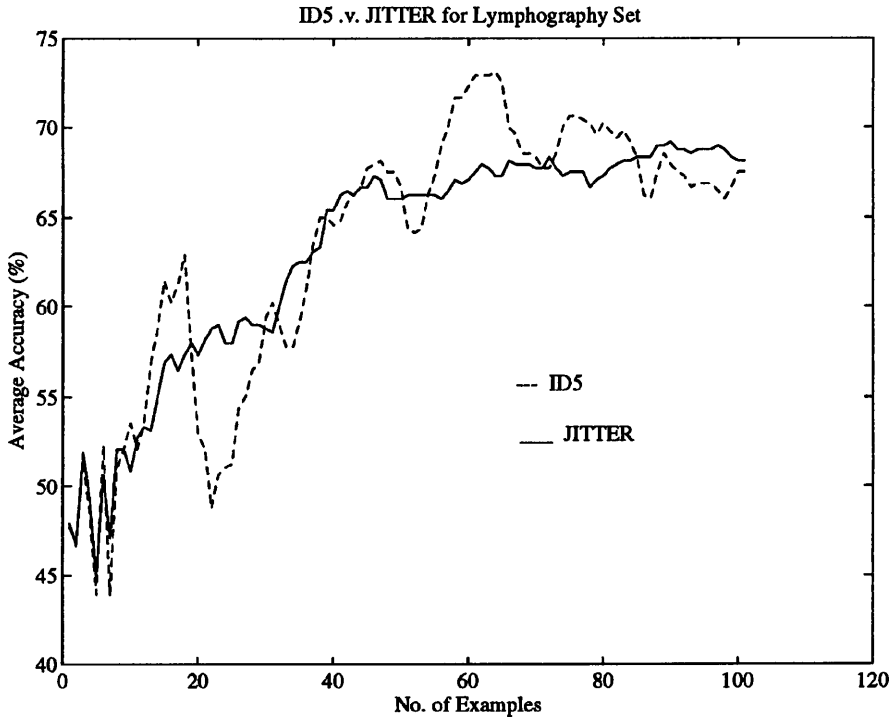



Figure 5.1:

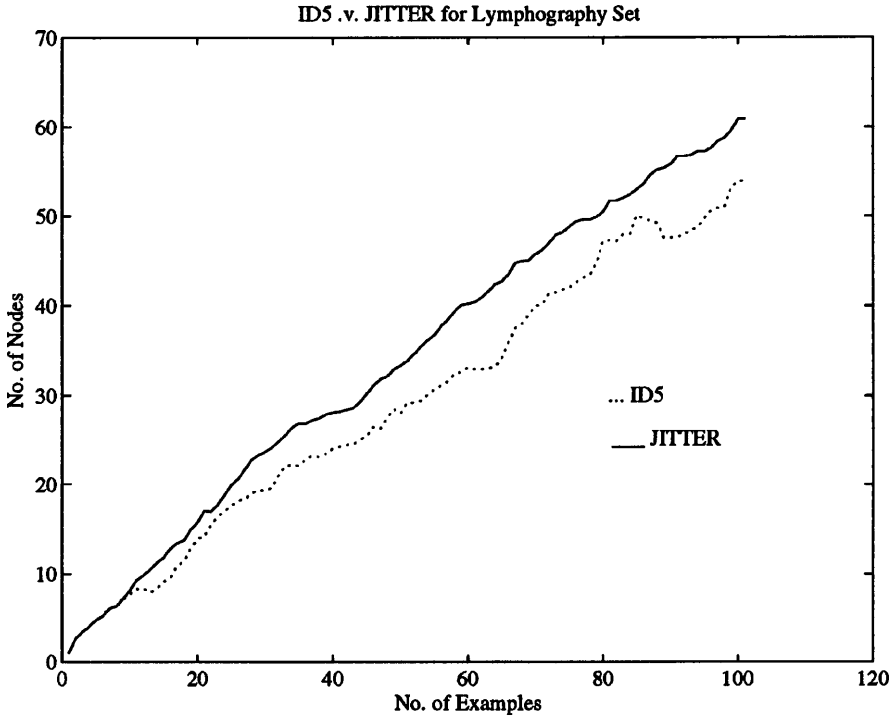



Figure 5.2:

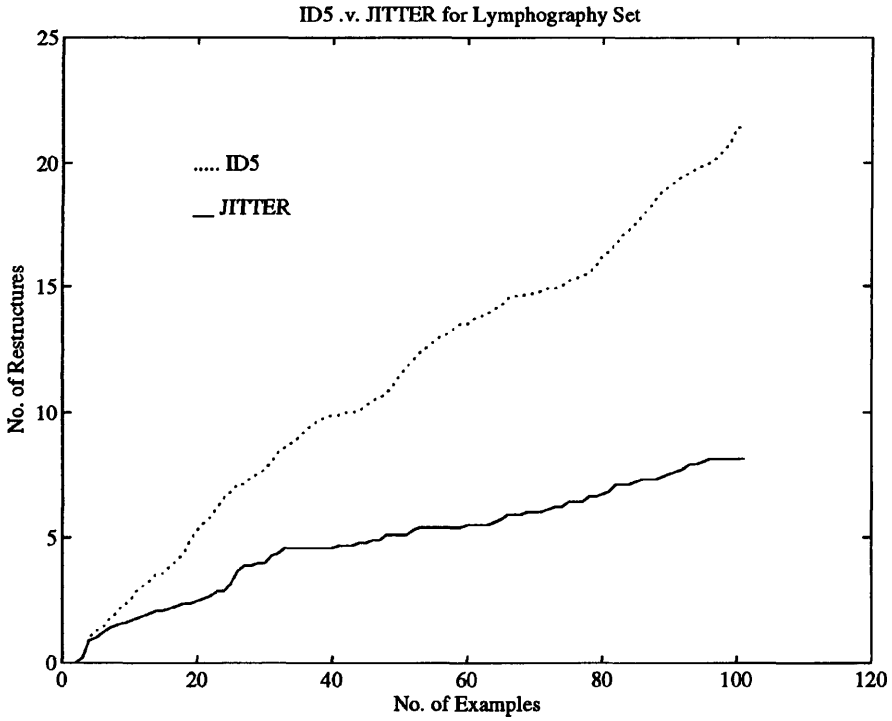



Figure 5.3:

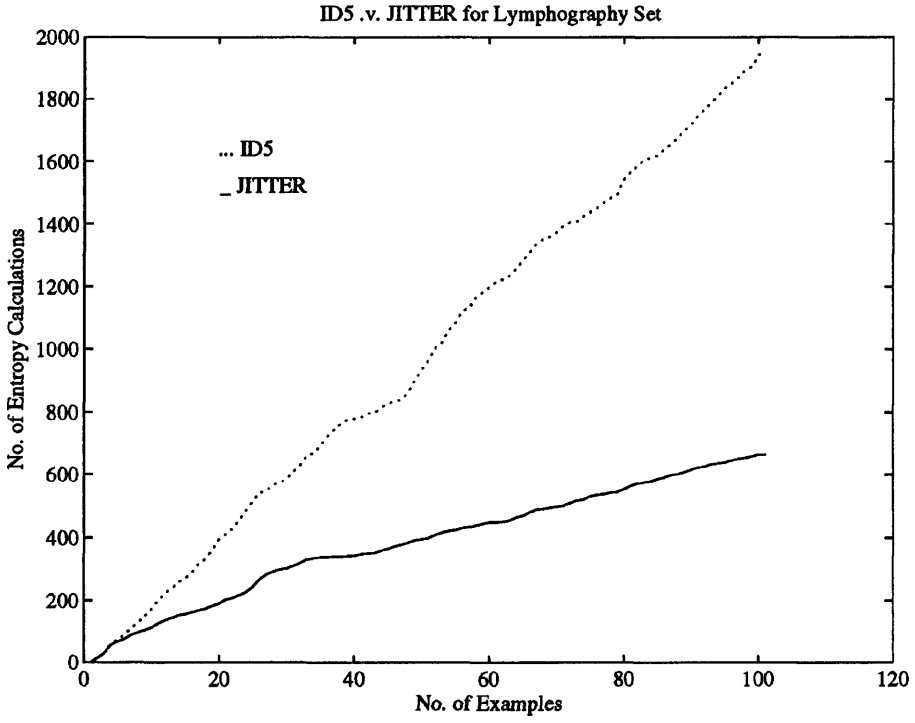



Figure 5.4:

5.2), although fluctuations appear smaller for JITTER. In fig. 5.3 JITTER's restructure counts are considerably less than those of $\widehat{\text{ID5}}$ and finally, in fig. 5.4 one can see the large differences in entropy calculation tallies.

## 5.4 Summary and Conclusions

From the above illustrations, and results presented in Appendix B, we conclude the following points. In §4.2 we mention a number of aims (viz., reduced run-time, number and extent of restructures, number of aborts and entropy calculations), whilst at least maintaining accuracy. To this end we identify a number of quantities in the JITTER model (§4.2.1)viz., the number of examples, entropy calculations, restructures, aborts, tree levels, node splits, node purity, accuracy, and restructure node counts. Here we briefly conclude what appears to affect these quantities and indeed if any relationships appear to hold between them.

It is axiomatic that the number of examples affects all these quantities. From the preceding graphs (figures 5.1, 5.2, 5.3, 5.4) one can see how accuracy, complexity, entropy calculations and restructures all change generally upwards as examples are added. It follows that aborts, splits and the total number of nodes restructured will do the same. In general we confirm that:-

1. increased cost parameters (e.g. numbers of entropy calculations) lead to increased run-times

2. increased restructures lead to increased entropy calculations, aborts and total nodes restructured, and thus increased run-time; however, it may also lead to *decreased complexity* and thus consequently *reduced* run-times, entropy calculations, tree levels and node splits, with increased accuracy and paradoxically perhaps decreased restructures, and so on (but this effect is not felt to be significant); ideally one must

optimise restructure counts in each domain, i.e. satisfy the complexity/accuracy versus inductive effort trade-off

3. increased 'aborts' lead to increased entropy calculations, and are especially wasteful

4. increased complexity leads to increased entropy calculations, and possibly decreased accuracy, increased likelihood of restructures, aborts and nodes restructured (see point (2) above); however, on the whole, it would appear that increased size leads to decreased accuracy, first and foremost (these other factors are less obvious)

5. increased node splits lead to increased complexity, and thus as (4)

6. increased node purity leads to decreased size and increased accuracy but often requires increased restructures, aborts and nodes restructured, entropy calculations and run-times

7. increased noise levels leads to increased restructures and aborts especially, and inductive costs in general (including size)

One can extrapolate for the converse cases. We believe these relations hold across all ID3, ID5 and JITTER family trees. We suspect such relations hold for other algorithms as well but have no evidence to substantiate such an observation.

In addition we conclude that there is a trade-off between restructures (and inductive effort in general) and accuracy/complexity. As restructures (etc.) decrease, a tree often appears to use node splitting to compensate, which leads to increased complexity. Hence one must still identify the *necessary* restructures. Tree building itself appears to be relatively cheap, whereas restructuring is expensive in terms of efficiency. In general, restructures are a significant expense but the size of these restructures perhaps seems less salient than the process of restructuring per se.

In §5.3.1 we compare the ID3 family of algorithms and concluded that incremental processing is computationally expensive, and that $\widehat{\text{ID5}}$ is the most efficient of these incremental algorithms. Perhaps more correctly, one should say that $\widehat{\text{ID5}}$ best optimises the trade-off between inductive efficiency and accuracy/complexity, but it is still some way behind ID3's efficiency. We also noted how ID3 tree-equivalence is often not necessary and may imply wasted effort.

In §5.3.2 we test the JITTER family, first in isolation and then together. Again the hat versions are most economical. We note how the selection of a suitable threshold is problematic and may even cause strange behaviour. ID5_DELAY and PSEUDO_JITTER greatly reduce inductive costs but often at the expense of increased complexity and decreased accuracy. This lends more evidence to the above conclusion that there is an inherent trade-off between inductive efficiency and concept quality, and that more selectivity of restructures is required.

In general, thresholds significantly reduce inductive effort and a small threshold may be beneficial across the board. These algorithms tend to reach plateaux (where R=0) and no other changes can then occur in cost parameters. This implies one should be careful; simply disallowing all restructures is not correct and one must set a threshold accordingly. Thus counter quantities differ in effectiveness with respect to efficiency, *and* the effort required to calculate and maintain them (e.g. PSEUDO_JITTER does not decrease EC counts as the threshold rises).

QUASI_JITTER requires the setting of a threshold plus a 'sensible' procedure for reducing the stringency of the accuracy criterion at higher levels in a tree. Omitting this favours restructures high up in a tree, as opposed to where they are *required*, leading to higher costs. Node purity is felt to be a useful measure of global quality (accuracy), and thus our *accuracy criterion* is thought to be a plausible model for restructure checking. QUASI_JITTER's costs increased with its node 'purity', and its use may well become

uneconomic for higher thresholds. The JITTER algorithms (J1, J2 and JTD) attempt to remove the expensive process of calculating purity at all nodes, and also consequently avoid the need to reduce the accuracy criterion as one climbs. However, on the whole QUASI_JITTER performs well and is felt to be significantly better than ID5_DELAY and PSEUDO_JITTER.

JITTER fares well in most domains, although its performance costs are sometimes greater than certain algorithms at *certain* thresholds, but it does not suffer from the need for user-defined parameters. In addition, JITTER's accuracy and complexity are occasionally better than ID5 and so it appears to be a good compromise choice for improving the above efficiency/quality trade-off. That is, it generally improves efficiency whilst on the whole maintaining accuracy.

In §5.3.3 we test our favoured algorithms (JITTER and QUASI_JITTER) against ID3 and ID5. It is obvious that ID5 wastes considerable computation and so JITTER and QUASI_JITTER can offer considerable gains in this respect. JITTER even allowed the use of a test set with which ID5 could not cope (§5.3.4). JITTER and QUASI_JITTER were often more comparable to the non-incremental ID3 with respect to a number of inductive costs. Indeed, our ID3 is 'one-shot' and does not make use of a "window" (the sets are not really large enough) which would extend run-times. Thus one can envisage scenarios where JITTER and QUASI_JITTER run quicker than ID3.

| ALG | B.1-B.58 | B.59-B.69 | B.1-B.69 |
|---|---|---|---|
| ID5 | 75.9 | 41.9, 47.9 | 58.9, 61.9 |
| QJ 2 | 72.6 | ? | ? |
| J1 | 74.6 | 52.0 | 63.3 |
| JTD | 74.5 | 52.3 | 63.4 |

Table 5.24: Average Accuracies over Test Table Ranges

On the negative side, JITTER and QUASI_JITTER can suffer from increased com-

plexity and occasionally decreased accuracy. However, if one averages accuracies for tables B.1-B.58 and B.1-B.69 one can see that there is very little difference between $\widehat{\text{ID5}}$, J1, JTD and $\widehat{\text{QJ}}(2)$ (table 5.24)[8] . The two values for $\widehat{\text{ID5}}$ are due to its failure for the 'Parity+10%' set, the first figure is inclusive (of zero) and the second is exclusive.

Noise may be a problem for J1 (although at present this is really only a 'feeling'), but one can add simple measures to deal with this (J2). In general, noise exacerbates inductive inefficiencies (e.g. $\widehat{\text{ID5}}$). Pruning does not appear to help JITTER's more complex trees (§5.3.5).

A basic analysis of intermediate quantities (§5.3.6) led us to conclude that JITTER reduces the size and frequency of restructures and the often 'wild' behaviour of $\widehat{\text{ID5}}$. It also confirms our thoughts that it is possible to reduce significantly actual and aborted restructures and entropy calculations, and thus costs in general, whilst maintaining tree quality (accuracy). In turn this leads to reduced volatility and instability, reduced risk of oscillation, and greater predictiveness of training behaviour.

[8] A crude measure but used by Quinlan 1993a. Note we omit the "noise" tables for the first average but include them for the second).

# Chapter 6

# Discussion

In Chapters 3, 4 and 5 we show how $\widehat{\text{ID5}}$ (and others) can waste significant effort during tree generation. This is mainly due to *unnecessary* and *aborted* tree restructures, which lead to unnecessary computation. In Chapters 4 and 5 we empirically demonstrate how several straightforward extensions can help to reduce 'effort' and thereby increase inductive efficiency. It will however be apparent that these algorithms offer advantages in some areas and not others. Below we discuss those results in more depth, and comment on the effectiveness and problems with our attempts to reduce inductive effort.

## 6.1 Analysis of Experiments

As previously noted, empirical evaluation of algorithms is not ideal, but often necessary where theoretical analysis is problematic. Obviously one can never be sure of the *general* effectiveness and robustness of an algorithm, but the more tests one undertakes the more positive one can be about any general conclusions. Extensive testing however is a time-consuming business and so one must strike a balance between 'completeness' and practical issues (e.g. the ease of analysis of results). We have chosen and used a reasonably wide range of both synthetic and real-world data sets, in order to evaluate our algorithms'

responses in general. To this end we believe we have shown what is possible with regard to efficient induction, and also the general behaviour of our algorithms. We believe these experiments pave the way for further, and more in-depth analysis and so feel reasonably confident as to the adequacy of our experimental procedure.

A number of further points are noted. The run-time value recording could benefit from a more rigorous procedure, e.g. averages over multiple runs, on dedicated machines. Example sets are randomly split into training and testing set pairs, ten times over. The use of 'leave-one-out' cross-validation would be worthwhile, if only to increase one's confidence in error rate estimates and counteract example ordering. The automatic tabling and graphing of the numerous inductive cost quantities (the dependent variables) would be a major help.

Some test domains display a remarkable resilience to increased complexity, i.e. accuracy does not suffer despite the size of trees increasing (e.g. chess KRKPA7). Other domains (e.g. LED) show resilience in complexity *and* accuracy, despite large differences in run costs etc.. We would suggest such domains are less useful for testing than those in which accuracy changes significantly with changes in complexity. To be of general applicability, an efficient algorithm must preserve quality in domains which are *not* relatively *complexity insensitive*, as well as those which are. Bohanec et al. (1994) notice a similar phenomenon when simplifying trees, i.e. many subtrees can be pruned with relatively little decrease in accuracy.

The efficiency criteria used to judge algorithms herein appear to be useful and well suited to the task. Run-time is more difficult to judge accurately than others due to the use of networked, multi-user computers. However, novel criteria such as the numbers of actual and aborted restructures, and nodes restructured, appear to give accurate empirical characterisations of the efficiency of learning. We would advocate the use of such parameters in the evaluation of algorithmic efficiency and thus extend the usual

methodology of using accuracy, size, and so on.

## 6.2 Analysis of Algorithms

There are often many trees which are functionally equivalent (i.e. of the same size and accuracy) for a particular concept. Our aim is ultimately, to build trees isomorphic to $\widehat{\text{ID5}}$ in less time, space, and so on[1] . Further improvement is necessary to achieve this ideal, but if one considers only accuracy, as we have done so far, then trees are mostly equivalent.

We wonder whether the restriction of restructures has the effect of pre-pruning, as 'outliers' have reduced chances of causing changes. In fact this would not appear to be the case, as only restructures and not tree growths are precluded, and as one can see, larger trees, not smaller, often result.

In general, if one requires ID3 tree-equivalence then the use of ID5R (Utgoff 1989b) would increase inductive effort still further. Indeed, we conjecture that in certain cases, ID5R would not produce a solution in 'reasonable' time and without requiring extensive resources (i.e. memory). That is, the number of restructures, entropy calculations, etc. will be higher still than for $\widehat{\text{ID5}}$. For all but the simplest of domains, our algorithms are unlikely to produce ID3-equivalent output. Does this present a disadvantage? Possibly, but as previously noted, ID3-equivalence is often not necessary or even desirable. Indeed, Breiman et al. (1984) hold that the tree generation procedure is not as relevant as the pruning techniques employed. In a number of our test cases, ID3 produces trees larger than $\widehat{\text{ID5}}$ and JITTER, and according to Van de Velde (1989, 1990), ID3-equivalence can entail convergence to suboptimal trees. On the whole though, we would suggest that in an incremental setting, *$\widehat{ID5}$ tree-equivalence* should be a general goal as it is *usually* an

[1] Two trees are isomorphic if one can map one into the other by permuting the order of the children of nodes.

accurate, complexity-sensitive, if relatively inefficient algorithm.

### 6.2.1 ID5_DELAY

As a simple extension to the ID5 algorithm, ID5_DELAY shows that considerable efficiency savings can be made. ID5_DELAY has a user-settable parameter to be initialised before each run (as have PSEUDO_JITTER and QUASI_JITTER). From the data used in any run, it is not obvious at what level the global threshold should be set. The only way to avoid this is either to make the threshold selection automatic or use one value for all tests. Thus at present we are left with the difficult choice of a compromise threshold value which appears to work reasonably well over many test sets.

For ID5_DELAY (and PSEUDO_JITTER) it is easy to set the global threshold so high as to disallow all restructures, which often leads to a large inaccurate tree. As the threshold starts low, it hinders restructures near the leaves, where fewest examples occur (*restructure requests* in PSEUDO_JITTER), precisely where one would expect most volatility to be observed. At first glance, this may seem intuitively appealing. However, as the threshold increases it has the effect of disallowing restructures further and further up the tree which we feel is intuitively wrong. That is, more and more examples (or restructure requests for PSEUDO_JITTER) will be required at a node at a given level, before a restructure is allowed. As the threshold increases, this only occurs in successively higher levels of a tree. For example, if the threshold is set at 100%, a restructure is only allowed at a node through which 100% of the examples have passed, i.e. the root. The effect is to *force* root-local changes when in reality, these levels should be becoming more stable. One should be restructuring further down *if possible*, not *insisting* on root-local changes. The ideal is to allow *necessary* changes at *any* level. From another angle, some *branches* may be 'correct' and some others may not, i.e. it may be better to restructure two children individually than restructure all on a level. It is difficult to see how a global

measure can be used as effectively in local decisions.

Top-down restructure checking still appears to be best in these cases (i.e. not bottom up) as the threshold stops activity near the leaves (i.e. bottom-up). One would therefore search many leaves unnecessarily if bottom-up checking were used.

In general, we believe the quantity on which ID5_DELAY's counter is based (i.e. the number of examples after a restructure is requested) is not ideal. Restructures should be allowed at any time, not simply 'banned' until a globally specified number of examples has been seen. Conversely, in noisy domains, where tree quality, as judged *during* training, will suffer regardless of the restructures made, it may well be beneficial simply to "wait and see", as then more evidence will be forthcoming. Greater flexibility in where and when to restructure is desired.

From our experiments, we have no reason to believe that ID5_DELAY's applicability across a range of domains is less than ID5's. Indeed, for 'sufficiently' high thresholds, the range of domains to which it is applicable is likely to be extended over ID5. On the down-side, its *utility* within domains is likely to decrease in comparison to ID5, as larger (i.e. less accurate) and more complex trees may well result. It would appear there is *not* a setting at which ID5_DELAY is equivalent to ID5. This is desirable if only for comparison purposes. As a threshold is relaxed, and if an ID5_DELAY tree 'tends towards' an $\widehat{\text{ID5}}$ tree, there may be a setting at which greater efficiency and $\widehat{\text{ID5}}$ tree-equivalence are both realised. At present, although greater efficiency is generally realised, $\widehat{\text{ID5}}$ tree-equivalence is almost definitely not. In conclusion we suggest $\widehat{\text{ID5}}$ is preferable for practical purposes, but that there exist domains in which its use is intractable and thus one must use a more efficient algorithm, such as ID5_DELAY. Quick, general learning may then form the basis for subsequent learning and refinement.

### 6.2.2 PSEUDO_JITTER

PSEUDO_JITTER also suffers from the non-automatic selection of its threshold, i.e. several runs may be required to determine an optimum which is counter to our aims of efficiency. As above, we have also seen how the threshold can have a deleterious effect on the *location* of a restructure, e.g. it is forced *up* a tree as the threshold increases.

Conceptually, one might think that the counter quantity for PSEUDO_JITTER would be better than for ID5_DELAY, i.e. the number of restructure requests is what is important, not necessarily the number of examples. In practice this amounts to an increase in the number of entropy calculations as one must recalculate for each example to see if a restructure is desired. Again this fairly crude counter quantity does not distinguish between *necessary* and *unnecessary* restructures but lumps them all together. Disallowing necessary restructures appears to leave trees at a disadvantage from which they struggle to recover. Perhaps if the frequency of restructures is fairly high, the effect is less obvious, but if the number of examples (and misclassifications) between restructures is high, the algorithm may use tree growths to counter decreasing accuracy. We surmise this leads to undesired stability in root-localities which hampers complete 'recovery' to $\widehat{\text{ID5}}$ equivalence. Perhaps an interim measure might be to allow growths only on example additions in which no restructure is requested.

The accuracy of PSEUDO_JITTER (and also ID5_DELAY) does not suffer too much in many domains, even when the threshold is at 100%. This warrants more investigation, but may imply that accuracy is less dependent on tree size in these domains (one could therefore elevate tree *size* in importance when considering tree quality). Is this counter to previous findings which state tree size is linked to error rate? We think not, as this only appears to occur in *complexity insensitive* domains. Thus one should take care with the sets used to evaluate an algorithm. If speed were of the utmost importance then

learning in such domains could tolerate very high thresholds, although in general one would prefer maximum clarity (i.e. minimum complexity, for a given accuracy).

As above, we see PSEUDO_JITTER's applicability extending beyond ID5's, but its utility decreasing. Compared to ID5_DELAY, one might prefer PSEUDO_JITTER, as it generally has quicker run-times, although its threshold is more stringent which leaves less leeway for experiment. We would also note that with a threshold of zero, $\widehat{\text{ID5}}$ equivalence *is* achieved for PSEUDO_JITTER.

### 6.2.3 QUASI_JITTER

This algorithm makes better use of information available from the example set during learning. That is, it makes use of one of the quantities associated with the functional equivalence of trees, viz. accuracy. It is difficult to compare ID5_DELAY, PSEUDO_JITTER and QUASI_JITTER directly as their counters are fundamentally different. Therefore we compare such things as run-times and node counts for the *maximum* and *minimum* accuracies recorded. For domains compared (e.g. mux, parity, soybean, LED, chess), QUASI$\widehat{\text{_JITTER}}$ more often than not gives smaller run-times and node counts. The best accuracy/complexity for ID5$\widehat{\text{_DELAY}}$ and PSEUDO$\widehat{\text{_JITTER}}$ is often with a threshold of zero, which is little different to $\widehat{\text{ID5}}$, if at all. QUASI$\widehat{\text{_JITTER}}$'s maximum accuracy and minimum complexity often exceeds PSEUDO$\widehat{\text{_JITTER}}$ and ID5$\widehat{\text{_DELAY}}$, and still saves much computation (e.g. soybean, parity, mux). In addition, PSEUDO$\widehat{\text{_JITTER}}$ and ID5$\widehat{\text{_DELAY}}$ have had more extensive testing (thresholds of 0%-100% in 5% or 10% steps), whereas QUASI$\widehat{\text{_JITTER}}$'s most 'economic range' ($1.0 < \textit{threshold} \leq 2.0$) is yet to be *fully* tested. Conversely, QUASI$\widehat{\text{_JITTER}}$ may be more susceptible to noise due to its reliance on single node purity in isolation. An additional heuristic based on tree size (although this is a global measure) may be of benefit in such a case. In hindsight, one could avoid accuracy criterion decrease problems by using only leaf accuracy as in

JITTER. However, which node on the return path to the root would one restructure? Perhaps one could utilise sub-tree accuracy, such that if it falls below global accuracy, one should restructure this sub-tree. Overall, QUASI_JITTER's modus operandi seems the most sensible so far, i.e. if accuracy is good enough, do not change anything. However, we would suggest one should still judge paths in their entirety, i.e. polythetic as opposed to monothetic appraisal. One 'bad' node in isolation might actually be very good in conjunction with the next one below.

If a large threshold is used, restructures occur everywhere, but these are checked bottom-up and so are more likely to be leaf-local. As the threshold decreases, accuracy at the leaves is accepted first and then progressively higher up. (Each child must have greater or equal purity than its parent, and so greatest purity occurs at the leaves.) Therefore, bottom-up restructure checks could be inefficient for lower thresholds, as restructures are more likely to be performed nearer the root. However, the accuracy criterion reduction mechanism significantly complicates matters. Perhaps the flexibility to perform top-down *or* bottom-up checks is required. This reflects our earlier observation on flexibility, i.e. an algorithm should pick the most expedient method, based on quantities of *quality and efficiency.*

We believe QUASI$\widehat{\text{_JITTER}}$ to be as *applicable* as the preceding two algorithms, but it also exhibits greater *utility* (i.e. accuracy and complexity improve). With regard to $\widehat{\text{ID5}}$, run-times are consistently and substantially reduced, but there are still a number of occasions where accuracy and complexity degrade for QUASI$\widehat{\text{_JITTER}}$ (but see table 5.24). Again, we would suggest there are domains in which QUASI_JITTER's use is *necessary*, and it is certainly to be preferred over ID5_DELAY and PSEUDO_JITTER.

### 6.2.4 JITTER

In contrast, JITTER makes more intuitive use of accuracy, i.e. judged at the leaves only, with success over time being a constituent part of this judgement. However this algorithm can also suffer from the need to stipulate a *certainty factor* generation procedure (and a weight update equation, although this is less problematic). One *can* alter the *cf* generation procedure across domains and as such JITTER appears little better than the previous algorithms as one may be tempted to select the most appropriate function. For JITTER a more sophisticated *cf* generation function might pay dividends, especially one which is more theoretically justifiable. Possibilities include a *cf* in terms of the underlying class distribution seen so far, and even 'error' calculations based on the standard deviation of node class distributions, as seen in Quinlan's M5 algorithm (1993a). Leaf accuracies should be described in terms of a priori class distributions, or better still, the class proportions seen so far. That is, for two classes, if one class accounts for 95% of the examples then a leaf with accuracy 95% might indicate its path is not achieving much. Another possibility is the automatic adjustment of a *cf* as the number of global correct/incorrect classifications alters, similar to the use of the number of aborted restructures in J2. Certainly a *set* of *cf* functions, or 'preference criteria' such as *lexical evaluation functionals* (Reinke et al. 1988) could be useful. However, one must guard against overly complex generation functions (the simpler the better).

The current *cf* generation function *could* be likened to resubstitution, i.e. the accuracy is based on the training data, not an independent test set. However, we do not judge the final accuracy by resubstitution but use such information only as a *guide* to restructuring during induction. We note that the reliance on "the last example to arrive" in calculating some *cf*s may be erroneous (§4.2.5). This should be beneficial if examples are temporally ordered, and especially if one wished to track concept drift. However, it may in fact

render JITTER more susceptible to example set ordering if examples are not temporally ordered, which is to be avoided. Suffice to say, further investigation is required here.

The use of weights and *cf*s appears to permit greater flexibility, i.e. admit the alteration of such counters *in situ*. That is, JITTER uses quantities which allow the influence and *adaptation* of *cf* generation (dynamic bias), for example, the number of actual and aborted restructures. Whatever the particular *cf* generation function used, the ability to adapt, especially in a context-sensitive manner (with respect to the training set) is to be encouraged. However one must not lose sight of *efficiency within the bounds of acceptable quality*, which must drive everything (cf. for example, Schlimmer et al.'s (1986a) and Aha's (1992) use of complex statistical calculations, although we have no knowledge of the consequences on run-time).

Lebowitz (1987) mentions trade-offs and parameters, and the need to optimise settings between mutually exclusive ideals. In some cases, e.g. the parity and mux concepts, the trade-off we have identified between accuracy/complexity and efficiency does not appear to hold. That is, JITTER improves accuracy, complexity and all efficiency parameters, over $\widehat{\text{ID5}}$. It is not altogether clear at present why JITTER has such marked success with the parity and multiplexor domains. Van de Velde (1989, 1990) notes that the 6-bit mux oscillates for ID5, i.e. a stable concept definition is never reached[2] . This may hold some clue, i.e. JITTER tends to stop large oscillations near the root and thus decrease the frequency and magnitude of change. It would appear that JITTER waits long enough to be able to judge several tests in sequence (polythetic) and thus accept a higher quality tree. JITTER's tendency to restructure near the leaves may also play a part.

Tadepalli notes (1989) that Lazy EBL results in over general explanations, whereas our 'lazy' trees can be over-specific, i.e. too many leaves. Lazy EBL produces less accurate concepts, in a possibly intractable situation (e.g. an imperfect theory), which

[2] for a 'continuous' training set

may be repaired later, if required. According to Van de Velde (1990), ID3 often finds suboptimal trees with respect to test redundancy and therefore size. It is obvious that our algorithms could benefit from reduced tree complexity in certain cases and thereby induce minimally complex, maximally accurate trees with the least of effort. As noted by Van de Velde (1990), the larger the tree, the greater the chance of test redundancy. So far, any improvement in tree complexity in our work has proved to be elusive. We conjecture that this occasional failing of say, JITTER is due to the sole reliance on accuracy as a means of deciding when to restructure. Thus a *cf* generation procedure which also entails complexity information could be beneficial (e.g. Van de Velde 1989, 1990). IDL chooses restructures by tree topology, ID5(R) chooses them via entropy whereas JITTER chooses them via entropy and leaf accuracy. Another possibility is that JITTER (J1 and J2) will restructure nearer to the leaves, possibly letting higher levels stabilise sooner than say, $\widehat{\text{ID5}}$. For the parity and multiplexor domains, where ID5 can cause oscillation, this appears to be good news, but less so for domains such as chess. JTD was used to counteract this and appears to work well in general, but does not do so well in the parity and mux domains. In the chess domain, JITTER's tree was larger, whilst in the parity and multiplexor domains it was smaller than $\widehat{\text{ID5}}$. In effect, chess 'examples' are generalisations themselves, i.e. they represent a number of positions. Reinke et al. (1988) state that this can cause problems in rules: GEM found it hard to generalise and the rules produced were very disjunctive. Each constituent complex was very conjunctive (i.e. specialised) and therefore difficult to generalise. This may also hold a clue as to JITTER's complexity problems in this domain, i.e. a wide tree with long paths which allows restructures near the leaves would be hampered in generalising further (i.e. restructuring nearer the root). Núñez (1991) uses an "incompleteness" measure which counts how many attribute values are 'covered' by a subtree. If insufficient numbers of values for the example subset in question are accounted for, the proposed generalisation

is rejected. It may be beneficial to adapt this to JITTER whereby it *stops* growing once a 'sufficient' number of values are covered in a subtree.

A more theoretical measure of functional equivalence of trees between ID5 and JITTER would be useful, i.e. give a graded, numeric estimate of equivalence, but is left for further work. In addition, some quantitative measure of the convergence and volatility/stability of a concept would be beneficial. Perhaps these could be measured by the average level at which restructures take place and the average stability of weights, respectively.

A number of questions need answering, one of which is, how many more or less examples does JITTER need to produce a functionally equivalent tree to ID5, and vice versa? As JITTER's (and QUASI_JITTER's) accuracy is less than $\widehat{\text{ID5}}$'s on *some* sets, it is safe to assume that JITTER/QUASI_JITTER would require more examples to reach the same level of accuracy. However, the converse is also true for some other sets. The need for further examples is counter to our aim of quick learning, and it may be that $\widehat{\text{ID5}}$ is quicker overall in these sets, if JITTER/QUASI_JITTER were to require *many* more examples. It is possible for the *intermediate* accuracy of a tree for JITTER (calculated after the addition of each example) to lag behind that calculated for $\widehat{\text{ID5}}$. In practice, accuracy can and does diminish with a restructure and thus it is just as likely (if not more so) that the intermediate accuracy for JITTER will be better than that of $\widehat{\text{ID5}}$ (e.g. see §3.2.1 and §4.2). The parity results at least, suggest that accuracy/complexity lag is not generally the case for JITTER. Further experimentation is required to judge algorithms' relative 'sample complexities'.

It would appear that the JITTER family is largely constrained by the performance of ID3, ID5 etc. with respect to tree quality. That is to say, it is not usual that JITTER will improve on ID5's accuracy or complexity counts (but see mux and parity results, and table 5.24). This is both intentional and a disappointment. Firstly, we have deliberately

tried not to change anything fundamental about the attribute selection procedure of ID3/5 as this was not our concern and has been dealt with elsewhere. This may more readily allow our ideas to be applied to different tree generation procedures, i.e. tree generation is a separate process to restructuring. Indeed, Donoho et al. (1995) make use of standard induction algorithms in order to avoid redesigning methods to deal with noise and so on. Secondly, we had hoped that the more informed approach would uncover some other useful information with which to improve on ID3.

In general, JITTER did better than $\widehat{\text{ID5}}$ in some domains, and worse in others (with regard to accuracy/complexity), but $\widehat{\text{ID5}}$ may be more robust in natural and/or noisy domains. Therefore we would suggest neither is superior with respect to utility within domains (e.g. see table 5.24). JITTER was parsimonious with resources and allowed a noisy parity set to be tackled. Therefore we suggest JITTER is applicable to a wider range of domains than $\widehat{\text{ID5}}$.

There is little to choose between J1 and JTD, and we have yet to try top-down restructure checking on QUASI_JITTER. J2 improves some sets substantially, but is worse in others. We would conclude that the information gleaned from the example set for J2 *is* useful but that more analysis is necessary in order to add robustness. As for J1 versus QUASI_$\widehat{\text{JITTER}}$, more often than not, the mid-range 'economic' threshold settings for QUASI_$\widehat{\text{JITTER}}$ ([1.0,2.0]) result in quicker, cheaper and occasionally more accurate runs. Thus it would appear that attempts to rid QUASI_JITTER of thresholds and unintuitive use of node purity almost seem counter-productive although overall, results are mixed. We would conclude JITTER is still not picking necessary restructures *only* and may also disallow some which are necessary. In general we would conclude that if complex and/or voluminous domains (e.g. mux and parity type problems) are present, our methods may prove *necessary*. Moreover, Seshu (1989) contends such "quasi-classical parity problems" do frequently occur in natural domains. Indeed, Quinlan (1993b) notes

how some medical problems, consisting of diagnostic tests, often require $j$ of the $k$ tests to be true before a positive result is recorded.

### JITTER's Other Parameters

A number of other algorithm variations and/or parameters were experimented on. These proved to be unfruitful and so have not figured in the final algorithms but are included for completeness. The many variations used in JITTER have helped to focus our work and to determine more worthwhile avenues. For instance, we have used the numbers of *correct* and *incorrect* classifications during training for generating certainty factors, and also the *information gain* from one level to the next. In order to judge whether tree complexity was unnecessarily high, we experimented with the maximum size per set, i.e. $\sum_{i=0}^{A} V^i$ nodes. For example, in the chess domain, $V^A$ is huge and so there is little variance in (e.g.) $\frac{no.nodes}{V^d}$ (where $d$ is the current number of tree levels). Other variations include:-

- the use of different values of threshold to compare against $cf$s (i.e. not zero), and indeed, the comparison with other weights
- the use of different starting weights, e.g. proportional to the level in the tree
- $cf$ decrease as it is propagated back up the tree (similar to QUASI_JITTER's accuracy criterion reduction)
- the use of one weight per node, instead of one per branch

## 6.3 Analysis of the JITTER Model

From our experiments it seems sensible to make use of the data in hand for the basis of any functional decisions, i.e. decisions on how to represent and change a concept

being learned. Ultimately, 'sensible' functional decisions will result in a 'sensible' final concept description. One option is to use the examples as a source of information. An alternative is to judge the concept description *in situ* as learning progresses, e.g. through cross-validation with a separate test set. This is obviously a computationally expensive option. To avoid this, one would require heuristic estimates of 'adequacy' e.g. size, clarity, modularity of rules, etc.. Such 'global' considerations may be inappropriately specified between different domains and/or changing user requirements. However, we believe such heuristics are still useful as generic guides, to be over-ruled by more appropriate, more specific, context-dependent information such as that provided by the data in use. Thus a combination approach which is more data-driven, as opposed to more model-driven, appears to be most useful in favourably affecting efficiency. (See also Rendell 1986.) Indeed, as pointed out in Chapter 2, some algorithms are data-driven, whilst others rely on a statistical model. We note how a statistical model can render an algorithm more robust as it is less reliant on single examples, whereas data-driven algorithms tend to be more focused. That is, an inaccurate model might 'suggest' many unfruitful hypotheses during its search for a concept description. We believe the JITTER model shows some benefits from *both* camps. The retention of entropy for test selection allows robustness whilst the use of *data-specified knowledge* adds focus to the concept manipulation stage. Van de Velde (1989, 1990) also uses examples to focus on "needy" tree parts and to drive change procedures. In an analogous situation, Quinlan (1993a) extols the virtues of combining instance-based and model-based methods to increase algorithm power. The IBL algorithm provides local information (prototypes) and the model provides global information, and thus both types of knowledge are used to improve class prediction. To this end, the JITTER model appears quite adequate. Note that data-specified knowledge is different to Buntine's *application-specific* knowledge or bias (1990), in that it is automatic and dynamic but possibly less focused and less effective in the presence of noise.

We identify a number of quantities which may be used to judge efficiency (e.g. the number of actual and aborted restructures, and entropy calculations) and then concept quality can be used to improve it. For example, we aim to reduce restructures *meaningfully*, and not simply 'blanket-ban' any change whatsoever. Thus we exploit information from the tree under construction, not merely the example set as used. In IDX, Norton (1989) uses $N$-ply lookahead to enhance test selection. Murthy et al. (1995) have found this to be of dubious benefit to tree quality and it significantly increases computation. If lookahead cannot bridge the gap between greedy (local) and optimal (global) solutions, then perhaps data-specified knowledge can.

As noted above, the information acquired during learning regarding weights, $cf$s etc. is context sensitive in that it is applicable to this data set only. We do not learn general knowledge, meta-knowledge, search heuristics etc., that may have some use in *some* domains and marginal use in others. This knowledge speeds learning in this domain and this domain only. Moreover it can be relatively quick to attain as there is little need for complex calculations. Quinlan (1990c) says trees are context-sensitive and so our use of context-sensitive information is well suited to this representation.

Any of the quantities identified in §4.2.1 could be used as the basis for a 'counter' for altering algorithmic behaviour, and in Chapter 5 we show a number of them at work. It is apparent that some offer more relevant information to an algorithm than others (for the purpose of influencing efficiency). In general, counter quantities should be easy to calculate and maintain and thus not increase the inductive costs of an algorithm in total. Some counters are 'global' (e.g. the number of examples seen so far), whereas others are local (e.g. node purity, leaf accuracy). The latter are more context sensitive, but then perhaps more prone to the vagaries of noise and missing values etc., and so the use of thresholds needs care. Can *global* thresholds be used effectively with local counters? It would appear so, for instance QUASI_JITTER uses a global 'purity' requirement and

applies this to individual nodes which are in effect instantaneous local counters.

The weights in JITTER are in effect a more complex equivalent of counters. Weights are affected by leaf purity over time, and by several leaves. Thus counters can also have a temporal element, i.e. be *instantaneous* (e.g. QUASI_JITTER) or *aggregated* (e.g. JITTER).

In terms of a general model, our concept-observation methods may be incorporated into any incremental induction procedure (perhaps with associated actions other than restructure reduction). One would thus render such an algorithm more knowledge-based and potentially reduce the often indiscriminate nature of heuristic learning algorithms. That is, we envisage a more involved use of information gleaned directly from the data set and thus ultimately reduce the need for biasing of the concept space altogether. Obviously one would also require a fast, efficient algorithm which could capitalise on the greater freedom allowed in the concept space. Relying on various biases (e.g. language, preference criteria) to constrain a (possibly NP-hard) search is not ideal and can have implications for the completeness of any solution. For example, Webb (1995) avoids the use of heuristic search in a machine learning problem and thus relaxes inherent bias. Use of dynamic feedback is beneficial (Schlimmer 1987b) and Cai et al. (1991) use example 'votes' to back-up rule 'usefulness' calculations. Núñez (1991) uses extra background knowledge in the form of *is-a* hierarchies, and attribute weighting.

Webb (1995) uses *admissible search algorithms* which are guaranteed to find a target or goal if one exists. This is achieved through efficient searching and pruning the search tree of unnecessary branches. The JITTER model can be thought of as performing search-space pruning through the *denial* of restructures at certain points. This is not as stringently enforced as in (Webb 1995) and so one might term it quasi-search-pruning. It is noted that JITTER (etc.) may in fact result in the replacement of one operator (transformation) with another (specialisation) and thus not prune the search space, but

alter it. However, the difference in magnitude of such substituted operations is not known, e.g. is a small growth less expensive than a large restructure? In general, this point may go some way to explaining JITTER's savings in memory and speed.

In terms of algorithmic speed, it is apparent that our algorithms are on the whole very much quicker than $\widehat{\text{ID5}}$. Increased decision function complexity can greatly increase run-time, e.g. Murthy et al. (1994). Here one can see the large increases in decision function complexity (oblique hyperplanes), e.g. find a threshold $t$ such that the split $A_i < t$ is optimum. The calculation of hyperplane coefficients, together with random perturbations to avoid local minima, and the possible production of more than one tree all impinge on efficiency. Murthy et al. state that not all node tests can be considered and that some pre-selection of training examples may be needed to circumscribe complexity. This becomes less necessary with more efficient learning.

Other identified sources of information with this model are worth noting. As we have shown it is possible to use an 'aborted' restructure to convey information to the algorithm. It also appears to us that the number of global classifications and misclassifications during the adding of new training examples to the tree offer possibilities along the same lines. This is not as accurate as the independent testing of a tree with the separate test set, but is a very much less time consuming (if somewhat optimistic) estimate of the tree's error rate which could be put to good use. This is a similar measure to QUASI_JITTER and JITTER except that it is a global as opposed to node-based measure. One could use this information to alter *cf*s in situ, i.e. increase/decrease according to relative concentrations of correct/incorrect classifications.

### 6.3.1 Comparisons With Other Paradigms

Aha (1992) states that IBL algorithms are relatively fast to learn, with low update costs, but relatively expensive storage and classification costs. In comparison, trees have lower

storage and classification costs, and JITTER improves both learning speed and storage *during learning*, although the sometimes larger resulting trees obviously require more storage. Aha also notes that IBL algorithms can be more sensitive to noise than trees. IBL algorithms demonstrate that greater speed is both realisable and desirable, and we have shown how this is possible.

One can see the analogous behaviour between reinforcement learning and TD($\lambda$) (§2.2.3), and especially, QUASI_JITTER and JITTER. That is, QUASI_JITTER uses intermediate node quality to judge, or predict, final tree quality *now* (without waiting for the actual outcome), in order to speed up learning. Thus might be construed as being analogous to TD(0), i.e. the use of the current state only to predict an outcome. JITTER might be thought of as similar to TD($\lambda$) where 'several' states are used to judge node quality, and only allow actions (restructures) if quality deteriorates. However, this is a loose analogy, as examples are not necessarily temporally ordered.

## 6.4 Analysis of Thesis Aims

We show that $\widehat{\text{ID5}}$ can waste significant effort in comparison to the JITTER family of algorithms. In §4.1 we identify a number of aspects which could benefit from improved efficiency and discuss these below.

Speed is a practically important issue (e.g. Sutton et al. 1993, Musick et al. 1993). Most, if not all test sets have been processed more quickly with our algorithms and are progressing towards the run-times of the non-incremental ID3. They are certainly substantially better than INC_ID3 and $\widehat{\text{ID5}}$. We conclude that we have accomplished this aspect easily and thus our algorithms are significantly more suitable for 'on-line' learning (for example), where speed is important. This enables the use of more examples, more complex examples, and possibly more complex decision functions and the general

relaxation of bias. Complex sets could be impossible for all but the largest of systems. Our results, especially with added noise, amply illustrate our algorithms' abilities regarding increasing data set size and complexity. Moving from smaller, simpler sets, to the larger, noisier, less well defined domains, one can see the growing margins of difference between efficiency parameters over $\widehat{\text{ID5}}$ (etc.). The larger the data set, the greater the margin, even for relatively uncomplicated concepts. It is also evident that given a certain absolute run time, our algorithms may process a significantly larger proportion of examples. Musick et al. (1993) use sub-sampling of the training set to best utilise large sets and improve efficiency. We reduce the need for such 'windows' and the associated risks and computation. This also demonstrates how the effects of increased decision function complexity may be ameliorated.

One can see (e.g. figure 5.3) how concept volatility and instability are significantly reduced. In some cases, this may occur more than it should and concept quality can decrease in parallel. Decreasing volatility also reduces the risk of oscillation of concept definitions, and increases the *predictiveness* of descriptions. That is, at time $t$, one can more reliably predict the form of a description at some later time $t+1$. Susceptibility to any one example is also reduced.

We have also shown how dynamic bias can be simple to calculate and easy to apply to good effect. We have not shown if one can *relax* bias as compared to ID5, but we are certain the JITTER family can cope with extra complexity including a more complex description and/or input language.

We show how relevant information may be gleaned from an example set and used to impart knowledge about training progress to an algorithm, and that it generally pays to delay restructuring decisions until further evidence is forthcoming. We therefore reduce the 'hypothesise-and-test' element of search and increase adherence to the Principal of Least Commitment.

Iba et al. (1988) say that memory requirements are often used to measure the practicality of a learning system. Shen (1992) also found $\widehat{\text{ID5R}}$ and ID5R to be inadequate in this respect, and Sutton et al. (1993) rue ID5's unbounded memory and computation requirements. We can empirically state that JITTER greatly improves on ID5 as far as memory requirements go (e.g. the parity example).

### 6.4.1 Summary

Therefore in conclusion, we note that incremental induction has been improved through the following points:-

- for incremental TDIDT, $\widehat{\text{ID5}}$ tree-equivalence is often a sensible aim, with regard to quality (accuracy and size), but not normally efficiency (compared to ID3)
- quick, general, but possibly less accurate learning may be necessary in some domains and applications (e.g. data mining), and some believe it to be worthwhile in any case (§2.5.1 and §4.1)
- the more explicit use of tree quality is a good basis for judging necessary changes
- tree quality is probably best judged as accuracy *and* complexity (for example), not either in isolation
- it is required to identify reliably the *necessary* restructures, and *imposing* limited volatility and/or instability is not ideal
- the 'wait-and-see' method of restructuring is generally beneficial, but the length of the 'wait' must be carefully judged
- maximum flexibility in where and when to restructure is desired, except in perhaps the largest of domains (where A is large), top-down restructure checks may be best; ideally, this should again be a dynamic choice

- dynamic alteration of behaviour and bias is ideal, but may present control problems, e.g. successively more abstract levels of bias or meta-knowledge may be required
- the use of data-specified knowledge, together with the further integration of data- and model-driven learning is beneficial and
  1. is efficient as it is quickly derived
  2. can reduce indiscriminate behaviour
  3. adds focus/context-sensitivity to relatively noise-tolerant algorithms
  4. needs careful use, e.g. regarding example ordering effects
  5. may help to close the gap between local and global optimality
  6. permits the dynamic alteration of bias and therefore reduces susceptibility to inappropriate bias
- the use of counters/weights and/or thresholds of varying granularity is a promising avenue and we show a diverse set in use
- the JITTER family is capable of considerable efficiency savings over $\widehat{\text{ID5}}$
- speed of execution is increased, to the extent that some run-times are more comparable to the non-incremental ID3
- a larger proportion of examples can be processed in a given time, or use of larger and/or more complex data sets, and/or relaxation of bias permitted
- concept volatility and instability are reduced, as is the risk of oscillation and the susceptibility to any one example
- more efficient use of time and memory have increased the applicability of algorithms through increasing the set of 'learnable' concepts

- increased restructure flexibility through the use of multi-phasic restructure checks

Moreover, our ideas do not directly impinge on fundamental tree building processes (e.g. attribute selection) and so may be combined with other incremental algorithms/paradigms. This would permit the integration of more robust noise handling protocols (for instance), and other efficiency measures, e.g. the cost-sensitive selection of attributes. The result is that we achieve our aims of improving incremental inductive efficiency whilst on the whole maintaining accuracy.

## 6.5 Towards a Theory of Inductive Concept Manipulation Efficiency

In this section we offer a framework characterisation of inductive concept manipulation, intended to facilitate study and discussion. Below we define terminology, assumptions, spaces and methodologies/algorithms.

We have shown in Chapter 5 that it is feasible to collect and use information from the training data. Specifically, one can glean relevant information on the quality and efficiency of learning from the example set, as it is accepted. Our central assumption (modified from before) states:-

**Assumption.** It is possible for all example sets and all inductive algorithms to collect useful information from the training set.

**NB.** If one included EBL (§2.3.3), which is mainly deductive, then training data is in relatively short supply, and we would be unsure of the efficacy of our methods.

**Assumption.** It is possible to manipulate concept descriptions efficiently over time, to produce a description of high quality (accuracy and complexity), and therefore that manipulations are worthwhile.

So what is "Concept Manipulation"? One can see from ID5 that restructures may result in the transformation, generalisation or specialisation of a concept. The usual change is a *transformation*, where node orders are permuted. Within such restructures, nodes may be pruned (removal of redundant nodes with a single child), or added (some paths do not contain a newly identified 'best' attribute to be pulled-up and so must be extended). The former generally occurs in the latter stages of a restructure and corresponds to *generalisation*, whilst expansion occurs at the start and corresponds to specialisation. Therefore "concept manipulation" includes all splitting/growing, pruning and transformation/restructuring actions. Altering the order of attributes in a tree constitutes a change of hypothesis, therefore allowing restructures can be thought of as changing between competing hypotheses. The following applies to the (incremental) TDIDT paradigm but we comment on its applicability to rules etc. afterwards.

One can determine a space of *concept manipulation identification methods* (i.e. how to identify when a restructure etc. may be required):-

- no choice
- random choice
- when misclassifications occur
- when concept quality decreases (e.g. complexity)
- when concept quality is likely to be improved (heuristic)

The last point includes occasions when an algorithm's 'model' (e.g. impurity functions such as entropy) indicates that change is required. One can see that these alternatives range between data-driven and model-driven methods.

Once a concept manipulation is thought to be required, one can identify a space of possible manipulations. Typically one will have identified an inadequate sub-concept, i.e.

an internal test node and its children. The set of possible manipulations is circumscribed by the following alternatives:-

- discard the sub-concept (and perhaps subsequently retrain, e.g. ID4)
- discard and rebuild immediately (e.g. CART extensions §3.1.5)
- retain the sub-concept but manipulate the order (e.g. ID5)
- manipulate the sub-concept, then prune part or all (e.g. IDL)[3]
- 'grow' the existing sub-concept further (e.g. node 'splits' as in ID3)
- swap the sub-concept for another 'similar' part elsewhere[4]

Previously we have alluded to the difference between necessary and unnecessary restructures and so we define these more precisely. Henceforth we refer to *restructures* as a subset of the possible *concept manipulations.*

**Definition 1.** An *unnecessary concept manipulation* is one which does not increase accuracy nor decrease complexity. Conversely, a *necessary concept manipulation* is one which increases accuracy *and* decreases complexity. A *marginal* concept manipulation is one which either increases accuracy *or* decreases complexity[5] .

Thus in fact, most restructures are marginal manipulations. This new definition recognises the ideal situation of increasing accuracy *and* decreasing complexity simultaneously.

**Hypothesis 1.** An algorithm should strive to identify the necessary restructures, as cheaply as possible, and perform only these.

[3] pruning is led by a model or function, and is therefore not the same as 'discard' or 'discard-and-rebuild' types

[4] this last point is motivated by genetic algorithm search operators, e.g. cross-over

[5] alternatively one could say that a *necessary* restructure is one which if *not* done results in decreasing accuracy and/or increasing complexity

**Corollary.** *Aborted* concept manipulations (i.e. concept manipulations not committed to, or partially calculated) are a considerable source of wasted effort, and should be minimised.

This does *not* imply however that all restructures and manipulations should be committed to, where possible. If a suggested manipulation is deemed unworthy, then discarding it may be the most efficient choice, especially if forcing the change would require further training to correct. The whole idea revolves around the identification of necessary manipulations.

**Corollary.** Disallowing *all* concept manipulations is false economy and often leads to large, inaccurate concepts[6] . Indeed, simply *reducing* concept manipulations *indiscriminately* may lead to large inaccurate descriptions.

However, one can only know marginal and/or necessary concept manipulations in hindsight, once they have been executed, so one must find a reliable means of prediction. How can one identify a necessary manipulation? It is too expensive to collect resubstitution accuracy estimates of a concept (sub-)structure affected by the proposed change. Therefore one needs to use some form of heuristic, such as tree quality (e.g. via leaf purity); or perhaps assume that decreasing complexity will most likely lead to increased accuracy (there is some justification for this in the literature). Thus one would only commit to a change which decreases complexity. This is easily judged, but one would still need to actually *do* the manipulation to find out. Note that one would have to preclude node growing/splitting actions from this heuristic, i.e. it would apply to restructures only, otherwise one would end up with a singular-node tree.

**Hypothesis 2.** Concept updates based on misclassifications only can save considerable effort *if* relatively few misclassifications occur.

We would suggest that viewing an example set *holistically*, in terms of the numbers

[6] this applies to restructures mainly as disallowing initial tree *growth* is obviously fallacious

of correct *and* incorrect classifications is potentially more beneficial.

**Hypothesis 3.** The *usefulness* of a concept manipulation depends on the *magnitude* of change of the concept evaluation quality used (e.g. accuracy), and alters dynamically throughout training.

Depending on the margin between the current tree accuracy and a theoretical maximum given or estimated a priori, different sized manipulations may be most appropriate. For example, if tree accuracy is very much less than the maximum possible with the example set seen so far, then one might conclude that a major revision is in order, especially if early on in training[7] . Alternatively, an almost maximally accurate tree, where stability has steadily increased, should not ideally be subjected to wholesale change. This again emphasises the need for dynamic bias. In the main though, the usefulness of a manipulation will be a function of the effect on the need for further training. Does the decision to change/not change boost the algorithm's sample complexity? Does the concept description stabilise and converge to an acceptably accurate, simple hypothesis? If the answers are yes and/or no (respectively), the manipulation is likely to be of little value. However, judging usefulness in such terms is practically impossible and so we choose quantities more easily calculated, e.g. magnitude. One might use 'usefulness' as a basis for selecting between several possible restructures.

**Definition 2.** *Concept volatility* is defined as the frequency of change, and includes all specialisation, generalisation and transformation operations.

Volatility implies (large) fluctuations in tree size and/or accuracy, and therefore less predictiveness of a tree's quality at any one time. Marr (1982) states that one should not change unless required to.

**Corollary.** *Concept stability* is defined as the extent of changes, over time.

These quantities could be measured against averages so far, e.g. the number of changes

[7] how one judges this is another matter!

per example, and the number of nodes per change, respectively. If such averages did not decrease over time one might be able to conclude that the concept description was oscillating (unstable). *Volatility* could be approximated by $\frac{R + AR}{no.examples}$, i.e. the number of actual and aborted restructures per example, as these are likely to account for most concept manipulations.

The use of counters and thresholds offers one way to manipulate concept manipulation frequencies (etc.), and so significantly alter the volatility of a concept description, but some quantities on which these are based are problematic.

**Hypothesis 4.** Manipulation inhibition thresholds may reach plateaux, beyond which no further concept alteration will occur.

**Corollary.** Identification of this plateau for each set, per domain, may be necessary to allow sensible behaviour and, ideally, near linear control of inductive cost parameter magnitude profiles.

That is, a linear change in a threshold should ideally result in a linear change in the magnitude of one or more 'cost' parameters (e.g. the number of restructures)[8] . Threshold plateaux thus represent non-linearities. Failure to provide linear control may result in a lack of predictiveness of the behaviour of an algorithm, which may hamper investigatory applications. Once reached, plateaux result in zero restructures but simply banning all restructures is erroneous.

**Corollary.** Some counter quantities are computationally expensive to calculate and maintain.

**Definition 3.** Counter quantities can either be *local* or *global*, and may be *simple* or *complex*, and *instantaneous* or *aggregated.*

That is, a quantity may be based on local information, such as node purity or sub-

[8] a magnitude *profile* would consist of a graph of cost magnitudes for different thresholds, for the current set, e.g. a column in a table of Appendix B

concept accuracy, or on global information such as the number of restructures. A counter may have many direct influences (e.g. several leaf purities) or be based on one simple quantity in isolation. Finally, a counter may be calculated at one point in time, or aggregated over a time span.

**Hypothesis 5.** The use of statistical quantities derivable from the input data allows the context sensitive, (data-specified) knowledge-based guidance of an algorithm, i.e. one can attach semantics to quantities.

**Hypothesis 6.** The retention of information is of paramount importance and one must preserve as much information as is practicable during training.

That is, no post- or pre-pruning should be allowed, unless done in a virtual fashion (Utgoff 1995). Removing nodes whilst training may hamper learning (e.g. as in ID4's 'restructures').

**Hypothesis 7.** Concept quality and inductive efficiency provide good sources of information for the control of incremental induction.

**Corollary.** Purity is a good indicator of the need for change, but assumes that greater purity *can* be achieved.

That is, a set may have a large percentage of noise, in which case one should relax the requirement.

**Definition 4.** *Multi-phasic integration* occurs when example addition and tree reviewing occur in separate stages.

We would suggest a multi-phasic integration strategy offers most flexibility. That is, incorporate the new example into the existing concept description in its entirety, check the quality of the concept, perhaps propose a manipulation as necessary, check the validity of the proposed manipulation and alter as required. Such a methodology allows one the flexibility to choose alternative procedures at each stage. For instance, in terms of trees, incorporating an example first, then checking tree quality, enables one to

check for restructures either root-down, or leaf-up (as in JITTER). Utgoff's ITI algorithm (§3.1.4) also implements multi-phase example addition, facilitating a number of "training modes". A "lazy" mode allows one to add several examples and only update counts (etc.) when an up-to-date tree is *required for some purpose.*

**Hypothesis 8.** Some domains are *complexity resilient* and do not decrease in accuracy as concept complexity increases and thus are less useful as test sets.

**Hypothesis 9.** There is an inherent trade-off between inductive efficiency and concept quality.

For instance, run-time, numbers of actual and aborted restructures, numbers of nodes restructured and entropy calculations are often inversely proportional to concept accuracy (and vice versa for complexity), for complexity sensitive domains.

**Corollary** This trade-off differs between test sets, individual test set variations, and domains.

**Hypothesis 10.** An inherent trade-off exists between concept manipulation *size* and concept manipulation *frequency*, i.e. stability and volatility.

For example, as the extent of restructures tends to 0, the number of (unrestricted) restructures may tend to infinity. On the other hand, as the size of restructures tends to the maximum possible (the size of the tree), the number of restructures *may* tend to a small number, but not necessarily. Thus we tentatively suggest the frequency $f \propto \frac{1}{|R|}$, the size of restructures (in nodes). However, this should not imply that root-local restructures are to be preferred. On the contrary, all things being equal, one should prefer the smallest change possible.

### 6.5.1 Application to Other Representation Paradigms

So how does this characterisation relate to other concept description formalisms? Other major inductive paradigms include logic, rules and instance-based methods. Different

formalisms will have different costs associated with the different steps of learning, namely generalisation, specialisation and transformation. For example, one formalism may find restructures more expensive than for trees, in which case, reducing them is even more important. It is also worth noting how algorithms learn, e.g. bottom-up or top-down etc..

For rules, 'restructures' may apply to a subset of rules (and thus are analogous to tree restructures), or to individual terms (e.g. swap the $j$th and $j+1$th terms[9] ). Transformation can consist of altering the order of rule terms (alters order of match), the order of rules in the rule base, the splitting of one rule into two or more parts, the agglomeration of two or more rules into one, and so on. Generalisation consists of adding disjuncts, removing conjuncts etc. (see list in §2.5.7), and vice versa for specialisation. However, attaching counters to individual rule constituents may increase complexity unduly. Trees are a concise representation, effectively summarising the use of attributes in an example set, i.e. *one* occurrence of the root attribute, up to $V$ occurrences of the next, and so on. Nonetheless, all other aspects of this framework apply.

If learning is incremental in ILP (§2.5.7), then adjusting Prolog clauses (etc.) could be equivalent to the alteration of rule sets (and even theory revision §2.5.8). There will be added complexity over TDIDT as ILP uses subsets of Predicate Logic. However, Quinlan (1990c) notes that 'propositional' learning deals with large volumes of data due to the representation simplicity, and thus can exploit statistical properties. ILP does appear to be able to 'make do' with far less and so the use of counters and data-specified knowledge may again be suspect. As above, the remainder of our framework is still applicable.

For instance-based learning, restructuring would imply the alteration of the concept boundary in $N$-space. This may correspond to the removal of examples or the reclassifi-

[9] this affects the order of match in an execution system, and may also allow the compaction of the rule set

cation of examples in $N$-space. One may operate on a number of examples (a generalisation), or one in isolation. However, such manipulations will occur during learning only if generalisation has been performed (e.g. Salzberg 1990), which is not typically the case. The use of counters is again most applicable to such generalisations, but the remainder of the framework applies.

Thus overall, restructures *do* occur in all these (incremental) paradigms, together with the usual generalisation and specialisation of concept descriptions, and so our hypotheses would seem applicable. Concept manipulation efficiency is an important part of inductive learning. A more thorough investigation is required but one might pay heed to the following observations.

Our ideas on data-specified knowledge depend, to a certain extent, on 'sufficient' numbers of examples, the more the better, and probably offer most advantage with large sets. Thus using our algorithms to learn with few examples may not benefit efficiency as much as with a large set (cf. explanation-based learning), especially if many counters are maintained.

The method of judging concept quality may be problematic, e.g. does one use individual rule accuracy (say), or rule-set accuracy, or both? An inexpensive, efficient method is required.

The applicability of some points of our theory may depend on individual algorithms in use and requires more in-depth analysis. Design of future algorithms should nonetheless bear in mind explicit use of quality and efficiency.

### 6.5.2 A General Framework for the Use of Data-Specified Knowledge

The following is a general framework for the use of data-specified knowledge:-

- decide on the granularity of counters, e.g. value, attribute, conjunct, disjunct, global
- decide where to attach these counters, and how many, e.g. values, attributes, rules/leaves, etc.
- decide on counter quantity, e.g. accuracy, size, number of misclassifications, entropy, etc.
- decide on update strategy (increment and/or decrement), e.g. after each example, after $e$ examples, after a misclassification, etc.
- decide on 'decision strategy', e.g. compare against thresholds, other counters, weights, use heuristics, perform statistical calculations, etc.
- initialise counter(s)
- select appropriate learning algorithm
- repeat until end
- add example(s) to concept description
- update counter(s)
- use values in appropriate way, e.g. increased efficiency, accuracy, decreased size, pre-prune, post-prune, transform concept description, specialise, etc.

The last three items may well be intertwined, not separate. If one instantiates these various stages/decisions, one arrives at an algorithm which may use data-specified knowledge. For instance ID5_DELAY, PSEUDO_JITTER, QUASI_JITTER and JITTER use various counters of differing granularities to improve run-time, efficiency and to dynamically alter bias.

## 6.6 Summary

In conclusion, we discuss experimental procedures, algorithms and models, and their relative successes and failures. We identify and comment upon the advantages and disadvantages offered by our approaches and conclude that they do indeed offer substantial benefits for efficient incremental induction. We close the main body of our research by offering an incipient framework on inductive concept manipulation efficiency (including other paradigms) and its practical application. We envisage a time when heuristic learning algorithms will pay heed to some form of this framework in order to achieve efficient, real-time learning.

# Chapter 7

# Conclusions

## 7.1 Thesis Summary

This thesis investigates the efficiency of incremental induction of decision trees, and how this process might be enhanced, possibly through the use of alternative biases. This work is based on the premiss that, for Machine Learning to improve its penetration of mainstream software, it must address issues such as the *efficient* acquisition, organisation and refinement of complex knowledge. Below we briefly summarise our work before commenting on desired extensions, and finally concluding this thesis.

Chapter 2 introduces and reviews machine learning, and steadily progresses towards the TDIDT paradigm. On the way we review basic representation formalisms, abstraction levels, and symbolic and subsymbolic learning. We then proceed to supervised and unsupervised induction, and after describing the former in more detail, we introduce and compare the main paradigms. Following this we review a number of seminal algorithms before concentrating on the TDIDT formalism, which forms the basis of this thesis. In §2.7.7 and Chapter 3 we introduce incremental induction and give cogent reasons for its use. Perhaps the most forceful is the requirement that all *biological* learning be incremen-

tal. Others include the need for efficient updates, tracking concept drift, revisable reasoning, increasing parsimony and tractability, avoidance of combinatorial explosions, and bi-directional searching. For learning algorithms to be generally applicable, they must facilitate *adaptation* and not merely in the ways envisaged by programmers. Threaded throughout this is the need for speed, i.e. real-time, efficient learning.

In Chapters 2 and 3 we introduce the use of decision trees for supervised concept learning, and conclude that they offer a concise and relatively fast method of induction. We continue with explanations of previous incremental TDIDT algorithms, i.e. INC_ID3, ID4, ID5, IDL, ITI and their various 'flavours' e.g. $\widehat{\text{ID5}}$ and ID5R. In §3.2 we analyse ID3/4/5 in more depth with particular emphasis on the restructuring phase of learning. We identify a number of potential causes of inefficient induction, primarily to do with unnecessary concept manipulations. These are caused, at least in part, by the reliance on model-driven attribute selection, i.e. the use of entropy for judging concept quality. We conclude that ID4/5 do not make best use of concept quality (e.g. accuracy and complexity) and that these algorithms wander in a relatively uninformed manner through the concept space in search of solutions. Analysis of entropy and worst-case analyses of ID4 and ID5 helped confirm our thoughts, and also helped us identify a number of quantities related to efficiency and decision trees.

In Chapter 4 we explain further our thoughts on the need for efficient induction. We cite several areas likely to benefit from increased efficiency, e.g. speed (average incremental cost update must be less than starting non-incremental learning from scratch); the complexity and size of data sets (real world tasks will become ever more demanding); the need for increased evidence before committing to a change; concept volatility and stability (change when change is necessary, not before, identify a useful change, and converge to a stable, acceptable solution); algorithm generality (increasing the information available may lead to the relaxation of 'global' restrictions, bias and pre-processing); and

generally, the improved use of resources. The result is that these algorithms become more applicable in applications such as mega-induction (Musick et al. 1993), on-line learning (Sutton et al. 1993), and data mining (Holsheimer et al. 1994).

We then introduce a basic model for our algorithms, based on alternative sources of information easily available from the data in use. We show how this data can be used for acquiring simple, context-sensitive information, and to facilitate significant improvement in efficiency. Our model utilises various dimensions of concept quality and inductive efficiency to augment concept manipulation decisions and also to alter dynamically the behaviour of algorithms. In particular, we focus on concept accuracy as the primary source of information and the suitability of a concept. We identify four algorithms, of increasing complexity and diversity, as extensions of ID5.

ID5_DELAY uses a global threshold and local counters, based on the numbers of examples seen, to delay restructures. Once a restructure is requested, a user-defined number of examples is waited before a restructure is possibly executed. PSEUDO_JITTER's counter is very similar but is based on the number of requested restructures. QUASI_JITTER makes explicit use of concept quality in the form of node purity and continues to manipulate concepts until a desired level of purity is attained at each node. This purity requirement may be reduced for successively higher levels of a tree. JITTER uses ideas from neural networks to enable the use of leaf purity over time, and maintains weights to record changes in classification success. Correction factors are generated at the leaves and are used to update weights attached to branches. Correct classifications lead to increased weights and incorrect ones to reductions. Once a weight is less than or equal to zero, a restructure may be performed. A number of variations on this theme are identified. A worst-case analysis of these algorithms helped to appreciate the relative advantages and disadvantages of each, in relation to their forebear ID5. We describe in detail what these algorithms aim to achieve and how they improve on ID5.

In Chapter 5 we empirically evaluate numerous algorithms over a wide range of test sets, and adumbrate basic relationships between inductive cost criteria. $\widehat{\text{ID5}}$ proved to be the most efficient of the ID3 family but for *all* there was considerable scope for the loss of efficiency. In general, we show that incremental induction is a computationally intensive process. Testing the JITTER family in isolation and then with ID3 and $\widehat{\text{ID5}}$ identified a number of interesting questions and findings. For instance, we investigate the use of counters/weights and thresholds and their inherent characteristics. We illustrate the effects of indiscriminate concept manipulations, the trade-offs between efficiency and quality, and the size and frequency of restructures. Our use of data-specified knowledge opens up the possibility of adaptation during learning, and we show a number of avenues for this. Finally we show "intermediate" cost profiles for a representative set and perform some experimentation with noise and pruning.

In Chapter 6 we discuss the results in more general terms and offer an appraisal of our algorithms, model and aims in general. We conclude that the JITTER model and family improve on ID5 in a number of aspects. We then describe our thoughts and findings in more abstract terms in the form of a framework of concept manipulation efficiency. To conclude, we hint at its applicability to incremental induction in general, and its practical application.

Below we list a number of areas in which we would like to see improvements and/or further investigation.

## 7.2 Further Work

We would like to see more complex, in-depth analysis of all algorithms concerned. Greater precision is required in the worst-case analysis we perform in order to characterise more accurately the algorithms' comparative efficiency. Extension to average-case analysis

would also be beneficial. However, we would suggest that the precision desired would require *quantitative* estimates of the *required* number of restructures (etc.) in each test set. Once these can be calculated, our ultimate aims will be nearer to realisation, i.e. one would be much nearer to identifying the *necessary* restructures. This is what analysis should strive to attain.

Further testing of our algorithms is required. In general one should focus on complexity sensitive domains, although others should not be neglected. As previously noted, some measure of tree equivalence would be useful for analysis purposes. That is, one could judge how 'close' one tree is to another and perhaps measure the effort required to 'map' one into the other (e.g. achieve $\widehat{\text{ID5}}$, or JITTER equivalence).

We would advocate the use of sample complexity testing, i.e. presenting the training set *ad infinitum*, until a desired level of concept quality is acquired. This would give an indication of how much more training would be required by, say, JITTER, in a domain in which its accuracy suffers, to reach the level of $\widehat{\text{ID5}}$. This can give a good estimate of an algorithm's robustness (convergence and volatility/stability), and as noted, its efficiency. It would also give some indication of the impact of restructures on the need for further training. For example, does JITTER simply spread the same number of restructures as $\widehat{\text{ID5}}$ out over a longer period, or truly reduce them?

We would like to see the addition of protocols for dealing with missing values and continuous attributes. This would at least allow a greater choice of test set and may further show the need for efficient induction.

Perhaps better pruning and even virtual pruning would be worthwhile. For example, in less tractable domains, JITTER's use (for instance) can lead to quick, efficient learning but large, cumbersome concepts. The use of IDL's pruning mechanism (Van de Velde 1990) to minimise test redundancy and thus refine 'coarse' descriptions when a concept is required for execution, may pay dividends. This should occur in a *virtual* fashion (Utgoff

1995) so that no loss of information occurs, should training be subsequently resumed.

It might be possible to use the branch weights in JITTER to generate rules with attached confidence factors. Thus one could induce probabilistic concepts, e.g. with ID5, one could use the relative frequencies at leaves to attach confidence factors. It seems to us to be better to use some measure of accuracy, rather than merely frequency, e.g. leaf purity (perhaps weighted by leaf frequencies).

Overall we would like to see the utility of our algorithms increased within the domains where quality suffers. We expect this will only be forthcoming when more reliable identification of the *necessary* restructures is realised. The ultimate aim is trees equivalent to, or better than, $\widehat{\text{ID5}}$. More accurate identification of *where* to restructure, and which to choose if there is more than one, such that the need for further training is reduced, is the aim. On a more specific level, we would like to see further investigation of our algorithms focus on the following points:-

- discover why ID5_DELAY(0) is not equivalent to PSEUDO_JITTER(0) and $\widehat{\text{ID5}}$
- discover why trees generated by our algorithms find it hard to recover to $\widehat{\text{ID5}}$ equivalence when fewer restructures are done (i.e. why does entropy not pick the best attributes regardless of the current tree)
- try fully recursive restructures for our algorithms and see if $\widehat{\text{ID5}}$ equivalence is then forthcoming (i.e. emulate ID5R in *performing* restructures)
- further investigation of QUASI_JITTER, especially the accuracy criterion reduction mechanism, plus try out top-down restructure checks
- try a new algorithm which is a cross between QUASI_JITTER and JITTER and (in some way) uses *instantaneous* counters, based on leaf purity only
- calculate JITTER's *cf*s in terms of the class distributions so far

- further analysis of the ordering of example sets and the effects on *cf* generation
- investigate the use of statistical measures based on the training data, e.g. some estimation of the significance of a proposed restructure

Finally, we believe the validation and extension of our framework to other incremental paradigms, and the subsequent embodiment of efficiency considerations in algorithms in general, is a worthwhile aim.

## 7.3 Conclusion

Learning is a facet of intelligence which is necessary for 'survival'. If the resource requirements of learning exceed the constraints placed on a system, then learning becomes obsolete. For example, if an animal cannot adapt to its environment quickly enough it will starve or become prey. If an embedded learning algorithm cannot repair a system's knowledge base quickly enough, it cannot justify its existence. Similarly, if it runs out of memory, nothing is achieved and one must look for alternative methods. Therefore learning algorithms should ultimately emphasise the resources required for operation. These include execution time, memory, prior preparation of data by domain experts, transformation into suitable representations and so on. The less one needs to make assumptions or place constraints upon such facets, the greater the potential applicability of an algorithm. One should be aiming to relax constraints and restrictive assumptions. Following this route at present often leads to the increase of resource use, due to the increased complexity. The aim of this thesis has been to investigate the efficiency of one field of induction, namely incremental TDIDT. The goal has been to improve efficiency whilst maintaining or improving concept quality. To a certain extent, one can claim this paradigm to be representative of others (e.g. rules) and as such our findings may be applicable to all incremental induction. Specifically, we demonstrate that improvement

over $\widehat{\text{ID5}}$ is desirable, possible and realisable. Indeed we would go so far as to say it is necessary for some domains, and for the future advancement of learning in general. That is, ID5 (and others) do unnecessary computation whilst learning and so can become very time and memory hungry. These algorithms represent relatively simple methods which cannot cope with more complex problems (e.g. relations in ILP), and search in a relatively indiscriminate fashion for concept descriptions. The potential for vastly increased complexity of concepts is obvious. One must cope with noise, missing values, continuous attributes, non-independent attributes, domains requiring polythetic analysis, through to non-linear combinations of attributes in order to describe a concept accurately. The potential then is for an efficient algorithm to improve greatly the performance on more complex tasks.

To investigate efficiency we implement four algorithms of varying complexity to test our ideas. We demonstrate through the use of simple quantities that it is straightforward to improve the efficiency of learning. In acquiring contextual information from the task in hand, we believe we have improved and increased the sources of information available to an algorithm, increased the flexibility of an algorithm, and facilitated adaptation during learning. Increased context-sensitivity is achieved on domain, test set and sub-concept levels. Our algorithms are inherently model driven (i.e. entropy), but increase the use of data-driven information, i.e. what we refer to as "data-specified knowledge". In this manner, our algorithms make more explicit use of concept quality and become applicable to a wider range of problems, and thus afford a user more choice. For example, one can choose to run quicker, with more examples, more complex examples, more complex decision functions, etc., according to one's requirements. Thus we have shown how alternative sources of bias are available and usable, and can be put to effective use in improving learning efficiency. We show how it is possible to use this information to implement an accuracy heuristic which can reduce the automatically repetitive nature

of the ancestor algorithms. The potential is for an algorithm to reduce the deleterious effects of relying on greedy search.

More specifically, in most cases we have reduced the number of entropy calculations, the number and extent of actual and attempted restructures and algorithm run-time. This has often been achieved with no significant decrease in accuracy. Indeed, in one test domain, improvement has been forthcoming on all counts and JITTERś statistics are more comparable to the non-incremental ID3. We believe that the gains shown by the JITTER family illustrate the extended computational savings to be made over larger and/or more complex domains.

The results presented here should therefore prove significant for those interested in induction in less tractable domains and/or "on-line learning" (Sutton et al. 1993), where efficient, general learning is the order of the day. Indeed, applications such as Knowledge Discovery in Databases, where the use of parallel computers is sometimes necessary to process the vast amounts of data, will benefit from increased efficiency. Thus we conclude that, especially in situations where real-life data is in use, one should not only bear in mind the quality of the output, but how one can best achieve such output. Unlimited time and resources are commodities none of us has and therefore the use of a relatively expensive algorithm could become untenable.

# Bibliography


[Aha et al. 1991] Aha, D., Kibler, D. and Albert, M. (1991), Instance-Based Learning Algorithms, *Machine Learning Journal*, vol. 6, no. 1, pp. 37-66, Kluwer.

[Aha 1992] Aha, D. (1992), Tolerating Noisy, Irrelevant and Novel Attributes in Instance-Based Learning Algorithms, *International Journal of Man-Machine Studies*, vol. 36, no. 2, pp. 267-87, Academic Press.

[Angluin et al. 1983] Angluin, D. and Smith, C. (1983), Inductive Inference: Theory and Methods, *ACM Computing Surveys*, vol. 15, no. 3, September, pp. 237-70, ACM.

[Angluin et al. 1988] Angluin, D. and Laird, P. (1988), Learning from Noisy Examples, *Machine Learning Journal*, vol. 2, pp. 343-70, Kluwer.

[Arbab et al. 1988] Arbab, B, and Michie, D. (1988), Generating Expert Rules from Examples in PROLOG, *Machine Intelligence 11: Logic and the Acquisition of Knowledge*, Hayes, J., Michie, D. and Richards, J. (eds.), Clarendon Press.

[Bain et al. 1987] Bain, M. and Muggleton, S. (1987), Non-Monotonic Learning, *Machine Intelligence 12 : Towards An Automated Logic Of Human Thought*, Hayes, J., Michie, D. and Tyugu, E. (eds.), pp. 105-19, Clarendon Press.

[Bareiss et al. 1990] Bareiss, E., Porter, B. and Weir, C. (1990), PROTOS: An Exemplar-Based Learning Apprentice, *Machine Learning: An AI Approach*, vol. 3, Kodratoff, Y. and Michalski, R. (eds.), Morgan Kaufmann.

[Ben-David 1995] Ben-David, A. (1995), Monotonicity Maintenance in Information Theoretic Machine Learning Algorithms, *Machine Learning Journal*, vol. 19, pp. 29-43, Kluwer.

[Bhandaru et al. 1991] Bhandaru, M. and Murty, M. (1991), Incremental Learning from Examples Using HC-Expressions, *Pattern Recognition*, vol. 24, no. 4, pp. 273-82, Pergamon Press.

[Blum 1992] Blum, A. (1992), *Neural Networks in C++ - An Object-Oriented Framework for Building Connectionist Systems*, Wiley.

[Blythe 1988] Blythe, J. (1988), Constraining Search in a Hierarchical Discriminative Learning System, *Proceedings - ECAI 88*, pp. 378-83.

[Bohanec et al. 1994] Bohanec, M. and Bratko, I. (1994), Trading Accuracy for Simplicity in Decision Trees, *Machine Learning Journal*, vol. 15, no. 3, June, pp. 223-50, Kluwer.

[Breiman et al. 1984] Breiman, L., Friedman, J., Olshen, R. and Stone, C. (1984), *Classification And Regression Trees*, Wadsworth International Group.

[Buntine 1990] Buntine, W. (1990), Myths and Legends in Learning Classification Rules, *Proceedings, 8th National Conference on AI*, vol. 2, pp. 736-42, AAAI/MIT Press.

[Cai et al. 1991] Cai, Y., Cercone, N. and Han, J. (1991), Attribute Oriented Induction in Relational Databases, *Knowledge Discovery in Databases*, Piatetsky-Shapiro, G. and Frawley, W. (eds.), pp. 213-28, AAAI/MIT Press.

[Cameron-Jones et al. 1994] Cameron-Jones, R. and Quinlan, J. (1994), Efficient Top-Down Induction of Logic Programs, *SIGART Bulletin*, vol. 5, no. 1, pp. 33-42, ACM.

[Carbonell et al. 1983] Carbonell, J., Michalski, R., Mitchell, T. (1983), An Overview of Machine Learning, *Machine Learning: An AI Approach*, Michalski, R., Carbonell, J. and Mitchell, T. (eds.), vol. 1, Morgan Kaufmann.

[Carbonell 1989] Carbonell, J. (1989), Introduction: Paradigms for Machine Learning, *Artificial Intelligence*, vol. 40, no.s 1-3, pp. 1-9, Elsevier Science.

[Cestnik et al. 1987] Cestnik, B., Kononenko, I. and Bratko, I. (1987), ASSISTANT 86: A Knowledge Elicitation Tool for Sophisticated Users, I. Bratko and N. Lavrac (Eds.), *Progress in Machine Learning - Proceedings of the 2nd EWSL 1987*, pp. 31-45, Sigma Press.

[Cheng et al. 1988] Cheng, J., Fayyad, U., Irani, K. and Qian, Z. (1988), Improved Decision Trees: A Generalised Version of ID3, *Proceedings - 5th International Conference on Machine Learning*, Laird, J. (ed.), Morgan Kaufmann.

[Cichosz 1995] Cichosz, P. (1995), Truncating Temporal Differences: On the Efficient Implementation of TD($\lambda$) for Reinforcement Learning, *Journal of AI Research*, vol. 2, pp. 287-318, Morgan Kaufmann.

[Cios et al. 1992] Cios, K. and Liu, N. (1992), A Machine Learning Method for Generation of a Neural Network Architecture: A Continuous ID3 Algorithm, *IEEE Transactions on Neural Networks*, vol. 3, no. 2, March, pp. 280-90.

[Clark et al. 1987] Clark, P. and Niblett, T., (1987), Induction in Noisy Domains, I. Bratko and N. Lavrac (Eds.), *Progress in Machine Learning - Proceedings of the 2nd EWSL 1987*, pp. 11-30, Sigma Press.

[Clark et al. 1989] Clark, P. and Niblett, T., (1989), The CN2 Induction Algorithm, *Machine Learning Journal*, vol. 3, pp. 261-83, Kluwer.

[Cockett et al. 1989] Cockett, J. and Zhu, Y. (1989), A New Incremental Learning Technique For Decision Trees With Thresholds, *Applications Of Artificial Intelligence 7*, SPIE - International Society For Optical Engineering, vol. 1095, no. 2.

[Conroy et al. 1994] Conroy, G. and Dutton, D. (1994), JITTER: A Lazy Machine's Guide To Induction, *Technical Report, UMIST-COM-AI-94-4*, Dept. of Computation, UMIST, PO BOX 88, Manchester, M60 1QD, UK.

[Conroy et al. 1995] Conroy, G. and Dutton, D. (1995), The Effects of Noise on Efficient Incremental Induction (Extended Abstract), *Machine Learning: ECML-95, Proceedings 8th European Conference on Machine Learning*, Lavrac, N. and Wrobel, S. (eds.), pp. 275-8, Lecture Notes in AI Series, Springer Verlag.

[Crawford 1989] Crawford, S. (1989), Extensions to the CART Algorithm, *International Journal of Man-Machine Studies*, vol. 31, pp. 197-217, Academic Press.

[Dayan et al. 1994] Dayan, P. and Sejnowski, T. (1994), TD($\lambda$) Converges With Probability 1, *Machine Learning Journal*, vol. 14, pp. 295-301, Kluwer.

[Decaestecker 1989] Decaestecker, C., (1989), Incremental Concept Formation Via a Suitability Criterion, *Proceedings - Conference on Data Analysis, Learning Symbolic and Numeric Knowledge*, Diday, E. (ed.), pp. 435-42, Nova Science.

[Decaestecker 1991] Decaestecker, C., (1991), Incremental Classification: A Multidisciplinary Viewpoint, *Proceedings - Conference on Symbolic-Numeric Data Analysis and Learning*, Diday, E. and Lechevallier, Y. (eds.), pp. 283-95, Nova Science.

[Dietterich et al. 1983] Dietterich, T. and Michalski, R. (1983), A Comparative Review of Selected Methods for Learning from Examples, *Machine Learning: An AI Approach*, Michalski, R., Carbonell, J. and Mitchell, T. (eds.), vol. 1, pp. 41-81, Morgan Kaufmann.

[Dietterich 1986] Dietterich, T. (1986), Learning at the Knowledge Level, *Machine Learning Journal*, vol. 1, pp. 287-316, Kluwer.

[Donald 1994] Donald, J.H. (1994), Rule Induction - Machine Learning Techniques, *Computing and Control Engineering Journal*, vol. 5, no. 5, October, pp. 249-55, IEE.

[Donoho et al. 1995] Donoho, P. and Rendell, L. (1995), Representing and Restructuring Domain Theories: A Constructive Induction Approach, *Journal of AI Research*, vol. 2, pp. 411-46, Morgan Kaufmann.

[Duda et al. 1973] Duda, R. and Hart, P. (1973), Pattern Classification and Scene Analysis, *Wiley.*

[Elomaa et al. 1990] Elomaa, T. and Kivinen, J. (1990), On Inducing Topologically Minimal Decision Trees, *Proceedings - 2nd International IEEE Conference on Tools for AI*, USA, November, pp. 746-52.

[Fatti et al. 1982] Fatti, L., Hawkins, D. and Liefde Raath, E. (1982), Discriminant Analysis, *Topics in Applied Multivariate Analysis*, Hawkins, D. (ed.), pp. 1-71, Cambridge University Press.

[Fayyad et al. 1990] Fayyad, U. and Irani, K. (1990), What Should Be Minimised In A Decision Tree, *Proceedings - 8th National Conference on AI*, vol. 2, AAAI/MIT Press.

[Fayyad et al. 1992] Fayyad, U., and Irani, K. (1992), The Attribute Selection Problem In Decision Tree Generation, *Proceedings - 10th International Conference on AI*, July, vol. 1, pp 104-10, AAAI/MIT Press.

[Fikes et al. 1971] Fikes, R. and Nilsson, N. (1971), STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, *Artificial Intelligence*, vol. 2, pp. 189-208, North-Holland.

[Fisher 1987a] Fisher, D.H. (1987), Conceptual Clustering, Learning from Examples and Inference, *Proceedings, 4th International Workshop on Machine Learning*, Langley, P. (ed.), Morgan Kaufmann.

[Fisher 1987b] Fisher, D.H. (1987), Knowledge Acquisition Via Incremental Conceptual Clustering, *Machine Learning Journal*, vol. 2, pp. 139-72, Kluwer.

[Fisher et al. 1989] Fisher, D. and McKusick, K. (1989), An Empirical Comparison Of ID3 And Back-Propagation, *Proceedings, 11th IJCAI*, vol. 1, Morgan Kaufmann.

[Gams et al. 1987] Gams, M. and Lavrac, N. (1987), Review of Five Empirical Learning Systems Within a Proposed Schemata, *Progress in Machine Learning - Proceedings of EWSL87: 2nd European Working Session on Learning*, Bratko, I. and Lavrac, N. (eds.), pp. 46-66, Sigma Press.

[Gelfand et al. 1991] Gelfand, S. and Delp, E. (1991), On Tree Structured Classifiers, *Artificial Neural Networks and Statistical Pattern Recognition: Old and New Connections*, Sethi, I. and Jain, A. (eds.), pp. 51-70, Elsevier.

[Gemello et al. 1989] Gemello, R. and Mana, F. (1989), An Integrated Characterisation and Discrimination Scheme to Improve Learning Efficiency in Large Data Sets, *Proceedings - 11th IJCAI*, Sridharan, N. (ed.), vol. 1, Morgan Kaufmann.

[Gennari et al. 1989] Gennari, J., Langley, P. and Fisher, D. (1989), Models of Incremental Concept Formation, *Artificial Intelligence*, vol. 40, no.s 1-3, Carbonell, J. (ed.), Elsevier Science.

[Goldberg 1989] Goldberg, D. (1989), *Genetic Algorithms In Search, Optimisation, And Machine Learning*, Addison-Wesley.

[Haussler 1986] Haussler, D. (1986), Quantifying the Inductive Bias in Concept Learning, *Proceedings - 5th National Conference on AI*, pp. 485-9, Morgan Kaufmann.

[Hawkins et al. 1982] Hawkins, D. and Kass, G. (1982), Automatic Interaction Detection, *Topics in Applied Multivariate Analysis*, Hawkins, D. (ed.), pp. 269-302, Cambridge University Press.

[Hinton 1990] Hinton, G. (1990), Preface to the Special Issue on Connectionist Symbol Processing, *Artificial Intelligence*, vol. 46, no. 1-4, pp. 1-4, Elsevier.

[Hoel 1984] Hoel, P. (1984), *Introduction to Mathematical Statistics*, Wiley.

[Holsheimer et al. 1994] Holsheimer, A. and Siebes, A. (1994), Data Mining - The Search For Knowledge in Databases, *Technical Report CS-R9406*, CWI, PO BOX 94079, 1090 GB, Amsterdam, Netherlands.

[Holte 1989] Holte, R. (1989), Alternative Information Structures in Incremental Learning Systems, *Machine and Human Learning - Advances in European Research*, Kodratff, Y. and Hutchinson, A. (eds.), pp. 121-42, Kogan Page.

[Hyafil et al. 1976] Hyafil, L. and Rivest, R. (1976), Constructing Optimal Binary Decision Trees is NP-Complete, *Information Processing Letters*, vol. 5, no. 1, May, pp. 15-7.

[Iba et al. 1988] Iba, W., Wogulis, J., Langley, P. (1988), Trading Off Simplicity and Coverage in Incremental Concept Learning, *Proceedings - 5th International Conference on Machine Learning*, Laird, J. (ed.), Morgan Kaufmann.

[Kearns 1990] Kearns, M.J. (1990), *The Computational Complexity of Machine Learning*, MIT Press.

[Keller 1987] Keller, R. (1987), Concept Learning in Context, *Proceedings - 4th International Workshop on Machine Learning*, Langley, P. (ed.), Morgan Kaufmann.

[Kibler et al. 1988] Kibler, D. and Langley, P. (1988), Machine Learning as an Experimental Science, *EWSL88 - Proceedings of the 3rd European Working Session on Learning*, Sleeman, D. (ed.), pp. 81-92, Pitman.

[Kibler et al. 1990] Kibler, D. and Aha, D. (1990), Learning Representative Examplars of Concepts: An Initial Case Study, *Readings in Machine Learning*, Shavlik, J. and Dietterich, T. (eds.), (Originally 5th Int. Workshop on Machine Learning), Morgan Kaufmann.

[Kocabas 1991] Kocabas, S. (1991), A Review of Learning, *The Knowledge Engineering Review*, vol. 6, no. 3, pp. 195-222, Cambridge University Press.

[Kodratoff 1988] Kodratoff, Y. (1988), *Introduction to Machine Learning*, Pitman.

[Kodratoff et al. 1990] Kodratoff, Y. and Michalski, R. (eds.) (1990), *Machine Learning: An AI Approach*, vol. 3, Morgan Kaufmann.

[Kolodner 1993] Kolodner, J. (1993), *Case-Based Reasoning*, Kluwer.

[Kononenko et al. 1991] Kononenko, I. and Bratko, I. (1991), Information-Based Evaluation Criterion for Classifier's Performance, *Machine Learning Journal*, vol. 6, no. 1, pp. 67-80, Kluwer.

[Kubat et al. 1991] Kubat, M. and Pavlickova, J. (1991), System FLORA: Learning From Time-Varying Training Sets, *Proceedings, EWSL91 - European Working Session on Learning*, Kodratoff, Y. (ed.), Springer Verlag.

[Laird et al. 1984] Laird, J., Rosenbloom, P. and Newell, A. (1984), Towards Chunking as a General Learning Mechanism, *Proceedings - National Conference on AI*, pp. 188-92, Morgan Kaufmann.

[Langley 1987] Langley, P. (1987), Preface, *Proceedings - 4th International Workshop on Machine Learning*, Morgan Kaufmann.

[Langley et al. 1987] Langley, P., Gennari, J, and Iba, W. (1987), Hill-Climbing Theories of Learning, *Proceedings - 4th International Workshop on Machine Learning*, Langley, P. (ed.), pp. 312-23, Morgan Kaufmann.

[Lebowitz 1987] Lebowitz, M. (1987), Experiments With Incremental Concept Formation: UNIMEM, *Machine Learning Journal*, vol. 2, no. 2, pp. 103-38, Kluwer.

[Lebowitz 1988] Lebowitz, M. (1988), Deferred Commitment In UNIMEM: Waiting To Learn, *Proceedings, 5th International Conference on Machine Learning*, Laird, J. (ed.), pp. 80-6, Morgan Kaufmann.

[Lenat 1982] Lenat, D. (1982), The Nature of Heuristics, *Artificial Intelligence*, vol. 19, no. 2, pp. 189-249.

[Lenat 1983] Lenat, D. (1983), The Role of Heuristics in Learning by Discovery: Three Case Studies, *Machine Learning: An AI Approach*, Michalski, R., Carbonell, J. and Mitchell, T. (eds.), vol. 1, pp. 243-306, Morgan Kaufmann.

[Lopez de Mantaras 1991] Lopez de Mantaras, R. (1991), A Distance-Based Attribute Selection Measure for Decision Tree Induction, *Machine Learning Journal*, vol. 6, no. 1, Kluwer.

[Manago et al. 1991] Manago, M. and Kodratoff, Y. (1991), Induction of Decision Trees from Complex Structured Data, *Knowledge Discovery in Databases*, Piatetsky-Shapiro, G. and Frawley, W. (eds.), pp. 289-306, AAAI/MIT Press,

[Marr 1982] Marr, D., (1982), *Vision: A Computational Investigation Into The Human Representation And Processing Of Visual Information*, Freeman.

[Matheus 1990] Matheus, C. (1990), Feature Construction: An Analytic Framework and an Application to Decision Trees, *Ph.D. Thesis*, University of Illinois at Urbana-Champaign.

[Michalski et al. 1980] Michalski, R.S. and Chilausky, R.L. (1980), Learning by Being Told and Learning from Examples: An Experimental Comparison of the 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. 125-61.

[Michalski 1980] Michalski, R.S. (1980), Pattern Recognition As Rule-Guided Inductive Inference, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-2, no. 4, July, pp. 349-61.

[Michalski et al. 1983a] Michalski, R.S. and Stepp, R.E. (1983), Learning From Observation: Conceptual Clustering, *Machine Learning: An AI Approach*, Michalski, R., Carbonell, J. and Mitchell, T. (eds.), vol. 1, Morgan Kaufmann.

[Michalski et al. 1983b] Michalski, R., Carbonell, J. and Mitchell, T. (eds.) (1983), *Machine Learning: An AI Approach*, vol. 1, Morgan Kaufmann.

[Michalski 1983] Michalski, R.S. (1983), A Theory And Methodology Of Inductive Learning, *Machine Learning: An AI Approach*, Michalski, R., Carbonell, J. and Mitchell, T. (eds.), vol. 1, Morgan Kaufmann.

[Michalski et al. 1986a] Michalski, R.S., Mozetic, I., Hong, J. and Lavrac, N. (1986), The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains, *Proceedings, 5th National Conference on AI*, pp. 1041-5, Morgan Kaufmann.

[Michalski et al. 1986b] Michalski, R., Carbonell, J. and Mitchell, T. (eds.) (1986), *Machine Learning: An AI Approach*, vol. 2, Morgan Kaufmann.

[Michalski 1986] Michalski, R. (1986), Understanding the Nature of Learning, *Machine Learning: An AI Approach*, Michalski, R., Carbonell, J. and Mitchell, T. (eds.), vol. 2, Morgan Kaufmann.

[Michalski 1990] Michalski, R. (1990), Learning Flexible Concepts, Chapter 3, *Machine Learning: An AI Approach*, vol. 3, Kodratoff, Y. and Michalski, R. (eds.), Morgan Kaufmann.

[Michalski et al. 1994] Michalski, R. and Tecuci, G. (1994), *Machine Learning: A Multistrategy Approach*, vol. 4, Morgan Kaufmann.

[Michalski 1994] Michalski, R. (1994), Inferential Theory of Learning, *Machine Learning: A Multistrategy Approach*, Michalski, R. and Tecuci, G. (eds.), vol. 4, Morgan Kaufmann.

[Mingers 1989] Mingers, J. (1989), An Empirical Comparison of Selection Measures For Decision Tree Induction, *Machine Learning Journal*, vol. 3, pp. 319-42, Kluwer.

[Minton 1988] Minton, S. (1988), *Learning Search Control Knowledge - An Explanation Based Approach*, Kluwer Academic Pubs.

[Mitchell 1977] Mitchell, T. (1977), Version Spaces: A Candidate Elimination Approach To Rule Learning, *Proceedings, 5th IJCAI*, pp. 305-10.

[Mitchell 1982] Mitchell, T. (1982), Generalisation As Search, *Artificial Intelligence*, vol. 18, pp. 203-26.

[Mitchell et al. 1983] Mitchell, T., Utgoff, P. and Banerji, R. (1983), Learning By Experimentation: Acquiring and Refining Problem-Solving Heuristics, *Machine Learning: An AI Approach*, Michalski, R., Carbonell, J. and Mitchell, T. (eds.), vol. 1, Morgan Kaufmann.

[Mitchell et al. 1990] Mitchell, T., Keller, R. and Kedar-Cabelli, S. (1990), Explanation-Based Generalisation: A Unifying View, *reprinted, Readings in Machine Learning*, Shavlik, J. and Dietterich, T. (eds.), Morgan Kaufmann.

[Muggleton 1987] Muggleton, S. (1987), Structuring Knowledge By Asking Questions, *Progress In Machine Learning*, pp. 218-29, Bratko, I. and Lavrac, N. (eds.), Sigma Press.

[Muggleton 1988] Muggleton, S. (1988), A Strategy for Constructing New Predicates in First Order Logic, *Proceedings, EWSL88 - 3rd European Working Session on Learning*, Sleeman, D. (ed.), Pitman.

[Muggleton et al. 1988] Muggleton, S. and Buntine, W. (1988), Machine Invention of First-order Predicates by Inverting Resolution, *Proceedings, 5th International Conference on Machine Learning*, Ann Arbor, June, pp. 339-52, Morgan Kaufmann.

[Muggleton et al. 1992] Muggleton, S., Srinivasan, A. and Bain, M. (1992), Compression, Significance and Accuracy, *Proceedings, 9th International Workshop on Machine Learning*, pp. 338-47, July.

[Muggleton 1994] Muggleton, S. (1994), Inductive Logic Programming: Derivations, Successes and Shortcomings, *SIGART Bulletin*, vol. 5, no. 1, pp. 5-11, ACM.

[Murphy et al. 1994a] Murphy, P. and Aha, D. (1994), UCI Repository of Machine Learning Databases, *Machine-Readable Data Repository*, Irvine, CA: University of California, Department of Information and Computer Science.

[Murphy et al. 1994b] Murphy, P. and Pazzani, M. (1994), Exploring the Decision Forest: An Empirical Investigation of Occam's Razor in Decision Tree Induction, *Journal of AI Research*, vol. 1, pp. 257-75, Morgan Kaufmann.

[Murthy et al. 1994] Murthy, S., Kasif, S. and Salzberg, S. (1994), A System for Induction of Oblique Decision Trees, *Journal of AI Research*, vol. 2, pp. 1-32, Morgan Kaufmann.

[Murthy et al. 1995] Murthy, S. and Salzberg, S. (1995), Lookahead and Pathology in Decision Tree Induction, *unpublished manuscript.*

[Musick et al. 1993] Musick, R., Catlett, J. and Russel, S. (1993), Decision Theoretic Subsampling for Induction on Large Databases, *Proceedings, 10th International Conference on Machine Learning*, June, Morgan Kaufmann.

[Natarajan 1991] Natarajan, B. K., (1991), *Machine Learning: A Theoretical Approach*, Morgan Kaufmann.

[Niblett 1987] Niblett, T. (1987), Constructing Decision Trees in Noisy Domains, *Progress in Machine Learning - Proceedings of EWSL87: 2nd European Working Session on Learning*, Bratko, I. and Lavrac, N. (eds.), pp. 67-78, Sigma Press.

[Norton 1989] Norton, S. (1989), Generating Better Decision Trees, *Proceedings - 11th IJCAI*, vol. 1, pp. 800-5, Morgan Kaufmann.

[Núñez 1988] Núñez, M. (1988), Economic Induction: A Case Study, *Proceedings, EWSL88 - 3rd European Working Session On Learning*, Sleeman, D. (ed.), Pitman.

[Núñez 1991] Núñez, M. (1991), The Use of Background Knowledge in Decision Tree Induction, *Machine Learning Journal*, vol. 6, no. 3, pp. 231-50, Kluwer.

[Pagallo et al. 1989] Pagallo, G. and Haussler, D. (1989), Two Algorithms that Learn DNF by Discovering Relevant Features, *6th International Workshop on Machine Learning*, Segre, A. (ed.), pp. 119-23.

[Pagallo 1989] Pagallo, G. (1989), Learning DNF by Decision Trees, *Proceedings - 11th IJCAI*, Sridharan, N. (ed.), vol. 1, Morgan Kaufmann.

[Quinlan 1979] Quinlan, J., (1979), Discovering Rules By Induction From Large Collections of Examples, *Expert Systems In The Micro-Electronic Age*, D. Michie (ed.), Edinburgh University Press.

[Quinlan 1983] Quinlan, J., (1983), Learning Efficient Classification Procedures and their Application to Chess End Games, *Machine Learning: An Artificial Intelligence Approach*, Michalski, R., Carbonell, J. and Mitchell, T. (eds.), vol. 1, pp. 463-82, Morgan Kaufmann.

[Quinlan 1986a] Quinlan, J., (1986), Induction of Decision Trees, *Machine Learning Journal*, vol. 1, Kluwer.

[Quinlan 1986b] Quinlan, J., (1986), The Effect of Noise on Concept Learning, *Machine Learning: An AI Approach*, vol. 2, Michalski, R., Carbonell, J., Mitchell, T, (eds.), Morgan Kaufmann.

[Quinlan 1987a] Quinlan, J., (1987), Decision Trees As Probabilistic Classifiers, *Proceedings of the 4th International Workshop on Machine Learning*, Morgan Kaufmann.

[Quinlan 1987b] Quinlan, J. (1987), Simplifying Decision Trees, *International Journal Of Man-Machine Studies*, vol. 27, pp. 221-34, Academic Press.

[Quinlan 1987c] Quinlan, J. (1987), Generating Production Rules From Decision Trees, *Proceedings - 10th IJCAI*, McDermott, J. (ed.), pp. 304-7, Morgan Kaufmann.

[Quinlan 1988a] Quinlan, J. (1988), Decision Trees And Multi-Valued Attributes, *Machine Intelligence 11: Logic and the Acquisition of Knowledge*, Hayes, J., Michie, D. and Richards, J. (eds.), pp. 305-18, Clarendon Press.

[Quinlan 1988b] Quinlan, J. (1988), An Empirical Comparison of Genetic and Decision-Tree Classifiers, *Proceedings - 5th International Conference on Machine Learning*, pp. 135-41, Morgan Kaufmann.

[Quinlan et al. 1989] Quinlan, J. and Rivest, R. (1989), Inferring Decision Trees Using the Minimum Description Length Principle, *Information and Computation*, vol. 80, pp 227-48, Academic Press.

[Quinlan 1990a] Quinlan, J. (1990), Learning Logical Definitions from Relations, *Machine Learning Journal*, vol. 15, no. 3, pp. 239-66, Kluwer.

[Quinlan 1990b] Quinlan, J. (1990), Probabilistic Decision Trees, *Machine Learning: An AI Approach*, vol. 3, Kodratoff, Y. and Michalski, R. (eds.), Morgan Kaufmann.

[Quinlan 1990c] Quinlan, J. (1990), Decision Trees and Decision Making, *IEEE Transactions on Systems, Man and Cybernetics*, vol. 20, no. 2, March/April, pp. 339-46.

[Quinlan 1993a] Quinlan, J. (1993), Combining Instance-Based and Model-Based Learning, *Proceedings, 10th International Conference on Machine Learning*, pp. 236-43, Morgan Kaufmann.

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

[Reinke et al. 1988] Reinke, R. and Michalski, R. (1988), Incremental Learning of Concept Descriptions: A Method And Experimental Results, *Machine Intelligence 11: Logic and the Acquisition of Knowledge*, Hayes, J., Michie, D. and Richards, J. (eds.), pp. 263-88, Clarendon Press.

[Rendell 1986] Rendell, L. (1986), A General Framework for Induction and a Study of Selective Induction, *Machine Learning Journal*, vol. 1, pp. 177-226, Kluwer.

[Rendell 1987a] Rendell, L. (1987), Similarity-Based Learning and its Extensions, *Computational Intelligence*, vol. 3, no. 4, pp. 241-66, November.

[Rendell 1987b] Rendell, L. (1987), Representations and Models for Concept Learning, *Technical Report*, UIUCDCS-R-87-1324, University of Illinois, March.

[Rendell et al. 1987] Rendell, L., Seshu, R. and Tcheng, D., (1987), More Robust Concept Learning Using Dynamically Variable Bias, *Proceedings - 4th International Workshop on ML*, Langley, P. (ed.), pp. 66-78, Morgan Kaufmann.

[Ringland et al. 1988] Ringland, G. and Duce, D. (1988), *Approaches to Knowledge Representation - An Introduction*, Wiley.

[Rivest 1987] Rivest, R. (1987), Learning Decision Lists, *Machine Learning Journal*, vol. 2, pp. 229-46, Kluwer.

[Rumelhart et al. 1986] Rumelhart, D., McClelland, J. and the PDP Research Group, (1986), *Parallel Distributed Processing - Explorations in the Microstructure Of Cognition - volume 1: Foundations*, MIT Press.

[Safavian et al. 1991] Safavian, S. and Landgrebe, D. (1991), A Survey of Decision Tree Classifier Methodology, *IEEE Transactions on Systems, Man and Cybernetics*, vol. 21, no. 3, May/June, pp. 660-74.

[Salzberg 1990] Salzberg, S. (1990), *Learning with Nested Generalised Exemplars*, Kluwer Academic Publishers.

[Sankar et al. 1991] Sankar, A. and Mammone, R. (1991), Combining Neural Networks And Decision Trees, *Applications Of Artificial Neural Networks 2*, SPIE - International Society For Optical Engineering, vol. 1469, pp. 374-83.

[Schaffer 1991] Schaffer, C. (1991), When Does Overfitting Decrease Prediction Accuracy In Induced Decision Trees And Rule Sets, *Proceedings, EWSL91 - European Working Session On Learning*, Kodratoff, Y. (ed.), pp. 192-205, Springer-Verlag.

[Schlimmer et al. 1986a] Schlimmer, J.C. and Granger, R.H., (1986), Beyond Incremental Processing: Tracking Concept Drift, *Proceedings, 5th National Conference on AI*, Philadelphia, PA, pp. 496-501, Morgan Kaufmann.

[Schlimmer et al. 1986b] Schlimmer, J.C. and Granger, R.H., (1986), Incremental Learning From Noisy Data, *Machine Learning Journal*, vol. 1, pp. 317-54, Kluwer.

[Schlimmer et al. 1986c] Schlimmer, J.C. and Fisher, D., (1986), A Case Study Of Incremental Concept Learning, *Proceedings - 5th National Conference on AI*, pp. 496-501, Morgan Kaufmann.

[Schlimmer 1987a] Schlimmer, J.C. (1987), Incremental Adjustment of Representations for Learning, *Proceedings - 4th International Workshop on Machine Learning*, pp. 79-90.

[Schlimmer 1987b] Schlimmer, J.C. (1987), Learning and Representation Change, *Proceedings - 6th National Conference on AI*, Seattle, Washington, pp. 511-5.

[Schoenauer et al. 1990] Schoenauer, M. and Sebag, M. (1990), Incremental Learning of Rules and Meta-Rules, *Proceedings - 7th International Conference on Machine Learning* , Porter, B. and Mooney, R. (eds.), pp. 49-57.

[Sebag et al. 1991a] Sebag, M. and Schoenauer, M. (1991), Learning by Successive Approximations, *Proceedings - Symbolic-Numeric Data Analysis and Learning*, Diday, E. and Lechevallier, Y. (eds.), pp. 215-30, Nova Science.

[Sebag et al. 1991b] Sebag, M. and Schoenauer, M. (1991), Discovering Meta-Rules from Rules and Examples, *Proceedings - 11th International Conference on Expert Systems and their Applications*, vol. 1, pp. 205-16, France.

[Seshu 1989] Seshu, R. (1989), Solving The Parity Problem, *Proceedings - EWSL89, 4th European Working Session on Learning*, Marik, K. (ed.), pp. 263-71, Pitman.

[Sethi et al. 1982] Sethi, I. and Sarvarayudu (1982), Hierarchical Classifier Design Using Mutual Information, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-4, no.4, July, pp. 441-5.

[Sethi 1990] Sethi, I. (1990), Entropy Nets: From Decision Trees To Neural Networks, *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1605-13.

[Sethi 1991] Sethi, I. (1991), Decision Tree Performance Enhancement Using An Artificial Neural Network Implementation, *Artificial Neural Networks and Statistical Pattern Recognition: Old and New Connections*, Sethi, I. and Jain, A. (eds.), pp. 71-88, Elsevier.

[Shapiro 1987] Shapiro, A. D., (1987), *Structured Induction in Expert Systems*, Addison-Wesley.

[Shavlik et al. 1990] Shavlik, J. and Dietterich, T. (1990), General Aspects of Machine Learning, *Readings in Machine Learning*, Shavlik, J. and Dietterich, T. (eds.), pp. 1-10, Morgan Kaufmann.

[Shen 1992] Shen, W. (1992), Complementary Discrimination Learning With Decision Lists, *Proceedings, 10th National Conference on AI*, July, pp. 153-8, AAAI/MIT Press.

[Shifu et al. 1992] Shifu, C., Bin, C, Jingui, P. (1992), ICAS: An Incremental Concept Acquisition System Using Attribute-Based Description, *Journal of Computer Science and Technology*, vol. 7, no. 3, July, pp. 284-8.

[Shlien 1992] Shlien, S. (1992), Nonparametric Classification Using Matched Binary Decision Trees, *Pattern Recognition Letters*, vol. 13, no. 2, February, pp. 83-7, Elsevier.

[Sleeman 1994] Sleeman, D. (1994), Towards a Technology and a Science of Machine Learning, it AI Communications - European Journal on AI, vol. 7, no. 1, March 1994, pp. 29-38, European Coordinating Committee for AI.

[Spears et al. 1990] Spears, W. and De Jong, K. (1990), Using Genetic Algorithms for Supervised Concept Learning, *Proceedings - 2nd International IEEE Conference on Tools for AI*, pp. 335-41, IEEE Comp. Soc. Press.

[Sutton et al. 1993] Sutton, R. and Whitehead, S. (1993), Online Learning with Random Representations, *Proceedings, 10th International Conference on Machine Learning*, Morgan Kaufmann.

[Swain et al. 1977] Swain, P. and Hauska, H. (1977), The Decision Tree Classifier: Design and Potential, *IEEE Transactions on Geoscience and Electronics*, vol. GE-15, no.3, July, pp. 142-7.

[Tadepalli 1989] Tadepalli, P. (1989), Lazy Explanation-Based Learning: A Solution to the Intractable Theory Problem, *Proceedings - 11th International Joint Conference on AI*, vol. 1, pp. 694-700, Morgan Kaufmann.

[Tan et al. 1988] Tan, M. and Eshelman, L. (1988), Using Weighted Networks to Represent Classification Knowledge in Noisy Domains, *Proceedings - 5th International Conference on Machine Learning*, Laird, J. (ed.), pp. 121-34, Morgan Kaufmann.

[Tan et al. 1990] Tan, M. and Schlimmer, J. (1990), Two Case Studies in Cost-Sensitive Concept Acquisition, *Proceedings - 8th National Conference on AI*, vol. 2, AAAI/MIT Press.

[Thrun et al. 1991] Thrun, S., Bala, J., Bloedorn, E., Bratko, I., Cestnik, B., Cheng, J., De Jong, K., Džeroski, S., Fahlman, S., Fisher, D., Hamann, R., Kaufman, K., Keller, S., Kononenko, I., Kreuziger, J., Michalski, R., Mitchell, T., Pachowicz, P., Reich, Y., Vafaie, H., Van de Velde, W., Wenzel, W., Wnek, J., Zhang, J. (1991), The Monk's Problems: A Performance Comparison of Different Learning Algorithms, *Technical Report, Carnegie Mellon University, CMU-CS-91-197.*

[Towell et al. 1992] Towell, G. and Shavlik, J. (1992), Using Symbolic Learning To Improve Knowledge-Based Neural Networks, *Proceedings - 10th National Conference On AI*, pp. 177-82, MIT/AAAI Press.

[Towell et al. 1994] Towell, G. and Shavlik, J. (1994), Refining Symbolic Knowledge Using Neural Networks, *Machine Learning: A Multistrategy Approach*, Michalski, R. and Tecuci, G. (eds.), vol. 4, Morgan Kaufmann.

[Utgoff 1986] Utgoff, P., (1986), *Machine Learning of Inductive Bias*, Kluwer Academic Press.

[Utgoff 1988a] Utgoff, P., (1988), ID5: An Incremental ID3, *Proceedings of the 5th International Conference on Machine Learning*, Laird, J. (ed.), pp. 107-120, Morgan Kaufmann.

[Utgoff 1988b] Utgoff, P., (1988), Perceptron Trees: A Case Study in Hybrid Representations, *Proceedings - 7th National Conference on AI*, pp. 601-6, Morgan Kaufmann.

[Utgoff 1989a] Utgoff, P., (1989), Incremental Induction of Decision Trees, *Machine Learning Journal*, vol. 4, pp. 161-86, Kluwer Academic Publishers.

[Utgoff 1989b] Utgoff, P., (1989), Improved Training Via Incremental Learning, *Proceedings - 6th International Workshop On Machine Learning*, Segre, A. (ed.), pp. 362-5, Morgan Kaufmann.

[Utgoff et al. 1990] Utgoff, P. and Brodley, C. (1990), An Incremental Method for Finding Multivariate Splits for Decision Trees, *Proceedings - 7th International Conference on Machine Learning*, Porter, B. and Mooney, R. (eds.), pp. 58-65.

[Utgoff 1995] Utgoff, P. (1995), Decision Tree Induction Based on Efficient Tree Restructuring, *Technical Report 95-18*, Dept. Computer Science, University of Massachusetts, Amherst, MA 01003.

[Vafaie et al. 1994] Vafaie, H. and De Jong, K. (1994), Improving a Rule Induction System Using Genetic Algorithms, *Machine Learning: A Multistrategy Approach*, vol. 4, Michalski, R. and Tecuci, G. (eds.), pp. 453-69, Morgan Kaufmann.

[Valiant 1984] Valiant, L., (1984), A Theory of the Learnable, *Communications of the ACM*, vol. 27, no. 11, pp. 1134-42.

[Van de Velde 1989] Van de Velde, W., (1989), IDL, or Taming the Multiplexor, *EWSL89 - Proceedings, 4th European Working Session on Learning*, Morik, K. (ed.), pp. 211-25, Pitman.

[Van de Velde 1990] Van de Velde, W., (1990), Incremental Induction of Topologically Minimal Trees, *Proceedings - 7th International Conference on Machine Learning*, pp. 66-74, Morgan Kaufmann.

[Vrain et al. 1988] Vrain, C. and Lu, C. (1988), An Analogical Method to do Incremental Learning of Concepts, *Proceedings, EWSL88 - 3rd European Working Session on Learning*, Sleeman, D. (ed.), Pitman.

[Wang et al. 1991] Wang, W. and Chen, J. (1991), Learning by Discovering Problem Solving Heuristics Through Experience, *IEEE Transactions on Knowledge and Data Engineering*, vol. 3, no. 4, December, pp. 415-9, IEEE.

[Watanabe et al. 1987] Watanabe, L. and Elio, R. (1987), Guiding Constructive Induction For Incremental Learning From Examples, *Proceedings - 10th IJCAI*, Italy, McDermott, J. (ed.), Morgan Kaufmann.

[Watson et al. 1994] Watson, I. and Marir, F. (1994), Case-Based Reasoning: A Review, *Knowledge Engineering Review*, vol. 9, no. 4, Cambridge University Press.

[Webb 1995] Webb, G., (1995), OPUS: An Efficient Admissible Algorithm for Unordered Search, *unpublished manuscript.*

[Weiss et al. 1991] Weiss, S. and Kulikowski, C. (1991), *Computer Systems That Learn - Classification And Prediction Methods From Statistics, Neural Nets, Machine Learning And Expert Systems*, Morgan Kaufmann.

[Winkelbauer et al. 1991] Winkelbauer, L. and Fedra, K. (1991), ALEX: Automatic Learning In Expert Systems, *Proceedings - 7th Conference on Artificial Intelligence Applications*, pp. 59-62, IEEE Computer Society Press.

[Wirth et al. 1988] Wirth, J. and Catlett, J. (1988), Experiments On The Costs And Benefits Of Windowing In ID3, *Proceedings - 5th International Conference on Machine Learning*, Laird, J. (ed.), Morgan Kaufmann.

[Wnek et al. 1994] Wnek, J. and Michalski, R. (1994), Comparing Symbolic and Subsymbolic Learning: Three Studies, *Machine Learning: A Multistrategy Approach*, Michalski, R. and Tecuci, G. (eds.), vol. 4, Morgan Kaufmann.

[Zadeh 1994] Zadeh, L. (1994), Fuzzy Logic, Neural Networks, and Soft Computing, *CACM*, March, vol. 37, no. 3, pp. 77-84, ACM Press.

[Zwitter et al.] Zwitter, M. and Soklic, M., (physicians), Institute of Oncology, University Medical Centre, Ljubljana, Yugoslavia.

# Appendix A

## A.1 Definitions

**kCNF** : all Boolean formulae: $P_1 \& P_2 ... \& P_j$ where each $P_i$ is a disjunction of at most k literals.

**kDNF** : all Boolean formulae: $P_1 \bigvee P_2 ... \bigvee P_j$ where each $P_i$ is a conjunction of at most k literals.

**k-clause CNF** : all Boolean formulae: $P_1 \& P_2 ... \& P_k$ where each $P_i$ is a disjunction of literals.

**k-clause DNF** : all Boolean formulae: $P_1 \bigvee P_2 ... \bigvee P_k$ where each $P_i$ is a conjunction of literals.

**Literal** : an atomic formula of the language.

**Horn clause** : a clause of at most one positive literal.

**Euclidean distance** : between two points, $x$, $y$, in $A$-space is calculated:-

$$Dist(x, y) = -\sqrt{\sum_{j=1}^{A}(x_j - y_j)^2}$$

**Normalised distance** : the absolute difference between two values, over the maximum range of the permissible values (summed over all attributes).

## A.2 Terminology

| | | | |
|---|---|---|---|
| **E** | the number of examples in total | **e** | a subset of E |
| **A** | the maximum number of attributes per example | **a** | a subset of A |
| **C** | the maximum number of classes in total | **c** | one class |
| **V** | the maximum number of values per example | **v** | a subset of V |
| **d** | the current level of a tree ($d \leq A$) | | |

Thus example spaces are of dimension $A$, trees (etc.) are of arity $V$, depth $A+1$, and so on.

## A.3 Statistical Testing

The following formula is used to calculate *t-test* values for determining the statistical significance of small samples.

$$\tau = \frac{\bar{x_1} - \bar{x_2}}{\sqrt{\left(\frac{\Sigma(x_1-\bar{x_1})^2+\Sigma(x_2-\bar{x_2})^2}{n_1+n_2-2}\right)\left(\frac{1}{n_1}+\frac{1}{n_2}\right)}}$$

where $\bar{x_i}$ is the mean for the $i$th sample ($\frac{\Sigma x_i}{n}$) and $n_i$ is the number of observations in the $i$th sample.

$\tau$ is thus a $t$-distribution value with $\nu = n_1 + n_2 - 2$ degrees of freedom (in our case, this is '18'). One then uses lookup tables for a desired $\alpha$ (= 95%, 99% or 99.9% confidence levels) and compares $\tau$ with the lookup value $\tau_{\alpha,\nu}$. If $\tau > \tau_{\alpha,\nu}$ then one can reject the 'null' hypothesis that $\bar{x_1} = \bar{x_2}$ and therefore conclude the two sample means are significantly different, with confidence $\alpha$.

# Appendix B

# Test Results

In the following tables, column headings signify: the algorithm in question (ALG); the time to build a tree in seconds (T); the number of nodes and leaves (NL); the number of leaves (L); the maximum path length (MP); the number of entropy calculations (EC); the number of restructures (R); the number of aborted restructures (AR); the number of nodes restructured in total (NR); and the accuracy in percent (AC)[1].

## B.1 Test Domains

Unless otherwise noted, all test sets have been obtained from Murphy et al. (1994a) via *anonymous ftp*[2] or the *World Wide Web*[3] .

### B.1.1 Balloons

This domain consists of 4 small data sets concerning the inflation of balloons. Each set has either 16 or 20 examples, with 4 binary valued attributes as follows: $COLOUR \in$

[1] Entries marked '?' are unavailable.

[2] ics.uci.edu, /pub/machine-learning-databases directory

[3] http://www.ics.uci.edu/AI/ML

*{yellow, purple}*, *SIZE* ∈ *{large, small}*, *ACT* ∈ *{stretch, dip}*, *AGE* ∈ *{adult, child}*. The class attribute *INFLATED* is either *True* or *False*. This domain is noise and missing value free. The following list gives the 'membership' function for the positive class in each set, i.e. *INFLATED* is *True* if (class distributions for *{True, False}* follow):-

1. age=adult OR act=stretch {12,8}
2. age=adult AND act=stretch {8,12}
3. (colour=yellow AND size = small) {8,12}
4. (colour=yellow AND size = small) OR (age=adult AND act=stretch) {7,9}

### B.1.2 Chess

#### KRKPA7

This set depicts the chess end-game of *king and rook versus king and pawn on A7*, abbreviated to *KRKPA7*. There are 3196 examples each of which has 36 binary attributes. The class depicts whether white (king and rook side) can win or not, with white to move (Shapiro 1987, Muggleton 1987). There are no missing values or noise and the classes are distributed as follows: win for white, 52%, and white cannot win, 48%. Each example represents a board description. The following is the list of features, in the order in which their values appear in the feature-value list: *[bkblk, bknwy, bkon8, bkona, bkspr, bkxbq, bkxcr, bkxwp, blxwp, bxqsq, cntxt, dsopp, dwipd, hdchk, katri, mulch, qxmsq, r2ar8, reskd, reskr, rimmx, rkxwp, rxmsq, simpl, skach, skewr, skrxp, spcop, stlmt, thrsk, wkcti, wkna8, wknck, wkovl, wkpos, wtoeg]*. One should consult the above references for further details.

### Knight Pin Chess End Game

In this set there are 1000 examples each with 22 attributes, and no noise or missing values. The class is decided according to whether the knight's side (black) loses in $n$-ply, where $n = 2, 3, ....$ The pieces consist of a king and knight (black), plus a king and rook (white). The 2-ply case is used here, i.e. a two move sequence may result in a loss for black. Each example is one end-game. Each attribute is typically ternary valued and the following list gives descriptions :-

- distance from black king to knight $\in \{1, 2, > 2\}$
- distance from black king to rook $\in \{1, 2, > 2\}$
- distance from black king to white king $\in \{1, 2, > 2\}$
- distance from white king to knight $\in \{1, 2, > 2\}$
- distance from white king to rook $\in \{1, 2, > 2\}$
- distance from rook to knight $\in \{1, 2, > 2\}$
- board relationship of black king and knight $\in \{diag, rect, other\}$
- board relationship of black king and rook $\in \{diag, rect, other\}$
- board relationship of black king and white king $\in \{diag, rect, other\}$
- board relationship of white king and knight $\in \{diag, rect, other\}$
- board relationship of white king and rook $\in \{diag, rect, other\}$
- board relationship of white rook and knight $\in \{diag, rect, other\}$
- type of black king's initial square $\in \{corner, edge, open\}$

- type of black knight's initial square $\in \{corner, edge, open\}$
- type of white king's initial square $\in \{corner, edge, open\}$
- type of white rook's initial square $\in \{corner, edge, open\}$
- rook checks black king $\in \{t, f\}$
- rook threatens knight $\in \{t, f\}$
- knight threatens rook $\in \{t, f\}$
- black king, knight, rook in line $\in \{t, f\}$
- black king can move adjacent to knight $\in \{t, f\}$
- knight can interpose adjacent to king $\in \{t, f\}$

This set was used by Quinlan (1983). The lisp program provided to generate the set produces duplicates and in our set (1000 examples), there were approximately 370 duplicates. The class distributions are as follows: lost (219), safe (781).

### B.1.3 Breast Cancer

This breast cancer data was originally provided by Zwitter et al., and has been used by Michalski et al. (1986a), Clark et al. (1987), and Cestnik et al. (1987). There are 286 examples, each of 9 attributes, some of which are linear, the rest categorical. There are 2 classes: *no-recurrence-events* (201), and *recurrence-events* (85). The following gives a list of the attributes and their values (see domain description - Murphy et al. 1994a) :-

- age: 10-19, 20-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80-89, 90-99.
- menopause: lt40, ge40, premeno.

- tumor-size: 0-4, 5-9, 10-14, 15-19, 20-24, 25-29, 30-34, 35-39, 40-44, 45-49, 50-54, 55-59.
- inv-nodes: 0-2, 3-5, 6-8, 9-11, 12-14, 15-17, 18-20, 21-23, 24-26, 27-29, 30-32, 33-35, 36-39.
- node-caps: yes, no.
- deg-malig: 1, 2, 3.
- breast: left, right.
- breast-quad: left-up, left-low, right-up, right-low, central.
- irradiat: yes, no.

There are some missing values, which for our purposes have been removed (i.e. the 9 examples are ignored).

### B.1.4 Cardio-Vascular

This set resulted from research carried out at the Renal Unit of Manchester Royal Infirmary and involved *Vascular Disease in Uraemia.* The study was intended to predict the likelihood of death from heart failure within four years of a kidney transplant. There are three classes :-

1. no heart failure within 4 years,
2. heart failure within 4 years,
3. heart failure and death within 4 years.

There are 27 attributes, such as age (discretised into 10 year ranges), sex, angina, diabetic, smoker, etc., as follows: [age (ordinal), mi (boolean), angina (boolean), cva (boolean),

lvf (boolean), pvd (boolean), male (boolean), hypertens (boolean), mht (boolean), renovasc (boolean), diabetes (boolean), smoke (boolean), hyperchol (boolean), hypertrig (boolean), anaemia (boolean), posfh (boolean), avfist (boolean), valvular (boolean), xrcalc (boolean), arrhyth (boolean), neph (boolean), pericard (boolean), hyperparath (boolean), pck (boolean), vasculitis (boolean), cardiomeg (boolean), abnecg (boolean)].

There are 305 examples, with no missing values and the distributions are as follows: (1) 218, (2) 21, (3) 66.

### B.1.5 Seven-Segment LED

This concept has 1000 examples, 10 classes ('0'-'9') and 7 Boolean attributes (indicating whether the LED segment is ON or OFF). The 'ON' attributes indicate an example's class. Noise levels range from 10% (i.e. each attribute has a 10% chance of being inverted), to 50%. This set has been used by Breiman et al. (1984), Quinlan (1987b), and Tan et al. (1988). There are no missing values, and class distribution is approximately 10%.

### B.1.6 Lenses

This set depicts the fitting of contact lenses. There are 24 examples, each with 4 nominal attributes. There are 3 classes corresponding to the fitting of *hard, soft*, or *no* contact lenses, to a patient. The set is complete (all possible attribute-value combinations are present), with no noise or missing values. The examples are highly simplified:-

> "The attributes do not fully describe all the factors affecting the decision as to which type, [of lens] if any, to fit." (Domain description, Murphy et al. 1994a).

The attributes are as follows:-

- age: young, pre-presbyopic, presbyopic
- spectacle prescription: myope, hypermetrope
- astigmatic: no, yes
- tear production rate: reduced, normal.

The class distribution is: *hard* (4), *soft* (5), and *none* (15).

### B.1.7 Lung Cancer

This set describes 3 types of lung cancer, with 32 examples of 56 nominal attributes (all values $\in \{0, 1, 2, 3\}$). There are 5 missing values, which again necessitates the removal of the offending examples. The class distribution is as follows: class 1 (9), class 2 (13), class 3(10). No other information is given on attribute descriptions (see Murphy et al. 1994a).

### B.1.8 Lymph Nodes

This lymphography data was originally provided by Zwitter et al., and has been used by Cestnik, et al. (1987), Clark, et al. (1987), and Michalski et al. (1986). There are 148 examples of 18 attributes each, no missing values and 4 classes. The attributes are as follows (Murphy et al. 1994a):-

- lymphatics: normal, arched, deformed, displaced
- block of affere: no, yes
- bl. of lymph. c: no, yes
- bl. of lymph. s: no, yes

- by pass: no, yes
- extravasates: no, yes
- regeneration of: no, yes
- early uptake in: no, yes
- lym.nodes dimin: 0-3
- lym.nodes enlar: 1-4
- changes in lym.: bean, oval, round
- defect in node: no, lacunar, lac. marginal, lac. central
- changes in node: no, lacunar, lac. margin, lac. central
- changes in stru: no, grainy, drop-like, coarse, diluted, reticular, stripped, faint,
- special forms: no, chalices, vesicles
- dislocation of: no, yes
- exclusion of no: no, yes
- no. of nodes in: 0-9, 10-19, 20-29, 30-39, 40-49, 50-59, 60-69, >=70

Class distribution is as follows: *normal find* (2), *metastases* (81), *malign lymph* (61), and *fibrosis* (4).

### B.1.9 Mux

Our next test set is the 11-bit multiplexor with 2048 examples, 8 data bits, 3 address bits and a class bit. The *mux* is used in narrow bandwidth communications to interleave

multiple messages to be passed along fewer lines. The address bits decide which of the data bits is to be passed through to the output. If the data bit indexed by the address bits is on (1), then the example is classed as positive. There is no noise (although we add some to further test sets) and no missing values.

### B.1.10 Noughts and Crosses

This set depicts a 'Tic-Tac-Toe' end-game. The examples describe all possible end-game configurations where 'x' is the first to play. There are some 958 examples, each with 9 attributes (one square on the board) and no missing values. The attributes are either $x$ (player 'x' has taken the square), $o$, or $b$ (blank) and there are 65.3% positive examples (win for 'x').

### B.1.11 Parity

The *even parity* concept consists of 2048 examples, each of 11 bits, only 5 of which are relevant (i.e. they decide the value of the final 12th, class bit). An example is classed as positive *iff* an even number of the relevant attributes is positive (i.e. '1'). There is no noise (although we add some to further sets), or missing values.

### B.1.12 Post-operative Care

This set classifies patients into 3 groups for post-operative care, viz. *intensive-care, hospital ward*, or *home*. There are 90 examples, of 8 attributes (plus class), as follows :-

- L-CORE (internal temperature in C): high (>37), mid ($\geq$ 36 and $\leq$ 37), low (<36),
- L-SURF (surface temperature in C): high (>36.5), mid ($\geq$ 36.5 and $\leq$ 35), low (<35),

- L-O2 (oxygen saturation in and <98), fair ($\geq$ 80 and <90), poor (<80),
- L-BP (last measurement of blood pressure): high (>130/90), mid ($\leq$ 130/90 and $\geq$ 90/70), low (<90/70),
- SURF-STBL (stability of surface temperature): stable, mod-stable, unstable,
- CORE-STBL (stability of core temperature): stable, mod-stable, unstable
- BP-STBL (stability of blood pressure): stable, mod-stable, unstable
- COMFORT (perceived comfort at discharge, an integer between 0 and 20).

There are some missing values (3 for attribute 8) and the corresponding examples have been removed for our purposes. The classes (and their distributions) are *I*: the patient should be kept in Intensive Care (2), *S*: the patient should be prepared to go home (24), and *A*: the patient should be returned to the hospital ward (64).

### B.1.13 Soybean

Michalski et al.'s classic induction problem (1980), but using a simplified data set (no missing values etc.). This set is derived from the larger, original data set (see Murphy et al. 1994a). There are 47 examples with 35 nominal attributes and 4 classes: *D1* (10), *D2* (10), *D3* (10) and *D4* (17), depicting diseases in soy plants. The full list of attributes and values is as follows :-

- date: april, may, june, july, august, september, october
- plant-stand: normal, lt-normal
- precip: lt-norm, norm, gt-norm
- temp: lt-norm, norm, gt-norm

- hail: yes, no
- crop-hist: diff-lst-year, same-lst-yr, same-lst-two-yrs, same-lst-sev-yrs
- area-damaged: scattered, low-areas, upper-areas, whole-field
- severity: minor, pot-severe, severe
- seed-tmt: none, fungicide, other
- germination: 90-100
- plant-growth: norm, abnorm
- leaves: norm, abnorm
- leafspots-halo: absent, yellow-halos, no-yellow-halos
- leafspots-marg: w-s-marg, no-w-s-marg, dna
- leafspot-size: lt-1/8, gt-1/8, dna
- leaf-shread: absent, present
- leaf-malf: absent, present
- leaf-mild: absent, upper-surf, lower-surf
- stem: norm, abnorm
- lodging: yes, no
- stem-cankers: absent, below-soil, above-soil, above-sec-nde
- canker-lesion: dna, brown, dk-brown-blk, tan
- fruiting-bodies: absent, present

- external decay: absent, firm-and-dry, watery
- mycelium: absent, present
- int-discolour: none, brown, black
- sclerotia: absent, present
- fruit-pods: norm, diseased, few-present, dna
- fruit spots: absent, coloured, brown-w/blk-specks, distort, dna
- seed: norm, abnorm
- mould-growth: absent, present
- seed-discolour: absent, present
- seed-size: norm, lt-norm
- shrivelling: absent, present
- roots: norm, rotted, galls-cysts

### B.1.14 The Monk's Problems

This set was used by Thrun et al. (1991) to compare many learning algorithms. There are 3 problems (one with noise) and each is divided into a training and a testing set (i.e. there are no random variations in this instance). There are 432 examples, 7 attributes (below), 2 classes $(0, 1)$ and no missing values.

- a1: 1, 2, 3
- a2: 1, 2, 3
- a3: 1, 2

- a4: 1, 2, 3
- a5: 1, 2, 3, 4
- a6: 1, 2
- Id: A unique symbol for each instance

The unique ID was removed for our purposes. The three concepts are as follows:-

- MONK1: (a1=a2) OR (a5=1)
- MONK2: Exactly two of {a1=1, a2=1, a3=1, a4=1, a5=1, a6=1}
- MONK3: (a5=3 AND a4=1) OR (a5 $\neq$ 4 AND a2 $\neq$ 3) plus 5% class noise added to the training set.

## B.2 Results

### B.2.1 Seven-Segment LED

[1]All algorithms mentioned herein, and other supporting code, is implemented in C under Unix on Sun Sparc ELC Workstations.

| | |
|---|---|
| **ID3** | see §2.6.4 and ff. |
| **ID4** | see §3.1.1, 3.2.1 and ff. |
| **ID5** | see §3.1.2, 3.2.1 and ff. |
| **DL xxx** | ID5_DELAY with an absolute or percentage threshold of *xxx* - see §4.2.2 |
| **PJ xxx** | PSEUDO_JITTER with an absolute or percentage threshold of *xxx* - see §4.2.3 |
| **QJ xxx** | QUASI_JITTER with a 'ratio' threshold of *xxx* : 1, correct : incorrect - see §4.2.4 |
| **QJ-TDEC** | QUASI_JITTER without the threshold decrease mechanism - §4.2.4 and §5.3.2 |
| **J1** | JITTER - see §4.2.5 and §5.3.2 |
| **J2** | JITTER with additions for handling noise - see §4.2.5 pp. 134, and §5.3.2 |
| **JTD** | JITTER with top-down restructure checking - see §4.2.5 and §5.3.2 |
| $\widehat{\text{DL}}$, $\widehat{\text{PJ}}$, **etc.** | 'hat' versions - update on misclassifications only |
| **ALG** | the algorithm in question (see also pp. 151 and 242) |
| **T** | the time to build a tree in seconds |
| **NL** | the number of nodes and leaves |
| **L** | the number of leaves |
| **MP** | the maximum path length |
| **EC** | the number of entropy calculations |
| **R** | the number of restructures |
| **AR** | the number of aborted restructures |
| **NR** | the number of nodes restructured in total |
| **AC** | the accuracy in percent |
| "?" | value unavailable, either due to algorithm failure (all missing) or algorithm does not produce all required figures |

Table B.0: Summary of Table Terminology

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| DL 0 | 118.4 | 158.9 | 77.2 | 8.0 | 17050.1 | 284.6 | 3274.8 | 12449.1 | 72.4 |
| DL 10 | 67.0 | 160.8 | 78.9 | 8.0 | 14339.4 | 227.1 | 2873.3 | 14229.1 | 72.4 |
| DL 20 | 57.1 | 162.2 | 79.8 | 8.0 | 12244.9 | 146.2 | 2493.6 | 11618.9 | 72.5 |
| DL 25 | 55.5 | 162.1 | 79.9 | 8.0 | 11414.3 | 125.8 | 2364.3 | 10311.3 | 72.7 |
| DL 30 | 52.2 | 161.0 | 79.3 | 8.0 | 10955.7 | 119.5 | 2288.0 | 8974.7 | 72.7 |
| DL 35 | 51.7 | 161.8 | 79.5 | 8.0 | 10436.6 | 88.0 | 2210.7 | 7321.2 | 72.6 |
| DL 40 | 47.6 | 161.9 | 79.4 | 8.0 | 10007.2 | 126.9 | 2151.4 | 12022.8 | 72.6 |
| DL 45 | 47.2 | 162.4 | 79.9 | 8.0 | 9545.2 | 65.9 | 2086.9 | 6457.5 | 72.6 |
| DL 50 | 46.8 | 162.6 | 79.8 | 8.0 | 9246.1 | 57.6 | 2033.5 | 5842.1 | 72.5 |
| DL 55 | 45.5 | 162.9 | 80.0 | 8.0 | 9254.4 | 82.8 | 2038.1 | 8334.5 | 72.6 |
| DL 60 | 52.5 | 162.1 | 79.7 | 8.0 | 8976.5 | 65.9 | 1983.5 | 6527.8 | 72.7 |
| DL 70 | 47.1 | 161.7 | 79.6 | 8.0 | 8417.4 | 48.5 | 1901.5 | 5242.0 | 72.7 |
| DL 75 | 45.2 | 161.2 | 79.6 | 8.0 | 8171.8 | 45.1 | 1862.1 | 4975.0 | 72.7 |
| DL 80 | 45.3 | 162.1 | 80.0 | 8.0 | 7980.4 | 31.5 | 1830.7 | 3308.2 | 72.5 |
| DL 90 | 42.7 | 162.1 | 80.0 | 8.0 | 7495.2 | 25.9 | 1765.3 | 2774.0 | 72.5 |
| DL 100 | 39.6 | 161.8 | 79.9 | 8.0 | 7162.8 | 71.3 | 1706.6 | 7368.4 | 72.5 |
| DL 125 | 41.1 | 162.0 | 79.8 | 8.0 | 6559.3 | 46.8 | 1620.8 | 4980.9 | 72.5 |
| DL 150 | 34.1 | 161.6 | 79.8 | 8.0 | 6551.6 | 83.0 | 1634.4 | 9589.8 | 72.7 |
| DL 175 | 28.7 | 162.1 | 80.1 | 8.0 | 5779.0 | 65.4 | 1519.2 | 7795.9 | 72.6 |
| DL 200 | 28.4 | 162.5 | 80.2 | 8.0 | 5427.7 | 54.9 | 1468.9 | 6823.3 | 72.5 |
| DL 250 | 23.7 | 162.0 | 80.0 | 8.0 | 4831.9 | 34.5 | 1363.1 | 4722.1 | 72.5 |
| DL 300 | 24.3 | 161.8 | 80.0 | 8.0 | 4362.9 | 17.6 | 1295.2 | 2744.1 | 72.5 |
| DL 350 | 26.2 | 161.9 | 80.1 | 8.0 | 4203.5 | 7.6 | 1259.8 | 1202.3 | 72.5 |
| DL 400 | 29.5 | 161.6 | 80.1 | 8.0 | 3774.8 | 0.8 | 1196.0 | 108.8 | 72.5 |
| DL 500 | 27.4 | 161.8 | 80.1 | 8.0 | 2968.5 | 0.6 | 1061.1 | 83.2 | 72.6 |

Table B.1: Results for 7-bit LED Set (10% noise) with Absolute Thresholds

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{DL}$ 0 | 68.9 | 154.1 | 75.1 | 8.0 | 5630.8 | 91.6 | 904.5 | 2176.4 | 72.6 |
| $\widehat{DL}$ 25 | 44.9 | 156.7 | 77.2 | 8.0 | 4206.4 | 44.9 | 719.1 | 3093.2 | 72.8 |
| $\widehat{DL}$ 50 | 41.8 | 159.4 | 78.2 | 8.0 | 3542.6 | 28.5 | 640.1 | 2509.1 | 72.8 |
| $\widehat{DL}$ 75 | 39.8 | 158.7 | 78.3 | 8.0 | 3092.8 | 12.3 | 574.7 | 1134.3 | 72.5 |
| $\widehat{DL}$ 100 | 38.2 | 158.7 | 78.4 | 8.0 | 2772.1 | 24.0 | 532.7 | 2294.9 | 72.4 |
| $\widehat{DL}$ 125 | 37.5 | 158.7 | 78.1 | 8.0 | 2598.1 | 16.4 | 516.6 | 1640.6 | 72.5 |
| $\widehat{DL}$ 150 | 31.7 | 158.7 | 78.3 | 8.0 | 2500.1 | 28.3 | 510.1 | 3138.4 | 72.7 |
| $\widehat{DL}$ 175 | 25.2 | 158.3 | 78.3 | 8.0 | 2236.7 | 21.1 | 476.0 | 2439.0 | 72.6 |
| $\widehat{DL}$ 200 | 25.9 | 159.6 | 78.8 | 8.0 | 2143.9 | 18.3 | 460.9 | 2214.3 | 72.6 |
| $\widehat{DL}$ 250 | 21.8 | 159.1 | 78.6 | 8.0 | 1949.2 | 9.9 | 431.5 | 1356.2 | 72.5 |
| $\widehat{DL}$ 300 | 23.5 | 159.1 | 78.6 | 8.0 | 1809.0 | 5.8 | 409.1 | 865.7 | 72.5 |
| $\widehat{DL}$ 350 | 25.0 | 159.1 | 78.7 | 8.0 | 1717.9 | 2.6 | 391.5 | 386.0 | 72.5 |
| $\widehat{DL}$ 400 | 29.2 | 159.0 | 78.8 | 8.0 | 1591.7 | 0.8 | 375.3 | 108.8 | 72.5 |
| $\widehat{DL}$ 500 | 24.5 | 159.8 | 79.0 | 8.0 | 1309.1 | 0.5 | 339.8 | 72.6 | 72.6 |

Table B.2: Results for 7-bit LED Set (10% noise) with Absolute Thresholds

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| DL 0 | 120.0 | 158.9 | 77.2 | 8.0 | 17050.1 | 284.6 | 3274.8 | 12449.1 | 72.4 |
| DL 5 | 56.6 | 162.2 | 80.0 | 8.0 | 13608.9 | 233.4 | 2705.6 | 17758.9 | 72.6 |
| DL 10 | 44.3 | 162.0 | 79.7 | 8.0 | 11737.1 | 171.2 | 2404.6 | 13859.8 | 72.7 |
| DL 15 | 39.6 | 162.4 | 80.0 | 8.0 | 10692.4 | 151.1 | 2263.4 | 14077.9 | 72.6 |
| DL 20 | 33.9 | 161.6 | 79.7 | 8.0 | 9595.2 | 119.1 | 2076.5 | 11392.2 | 72.3 |
| DL 25 | 32.0 | 163.0 | 80.3 | 8.0 | 8786.2 | 58.7 | 1959.5 | 4783.5 | 72.4 |
| DL 30 | 29.2 | 162.0 | 79.9 | 8.0 | 7673.7 | 34.1 | 1776.5 | 3199.5 | 72.4 |
| DL 35 | 26.3 | 162.5 | 80.2 | 8.0 | 7356.7 | 21.1 | 1730.3 | 1740.5 | 72.5 |
| DL 37 | 26.1 | 162.1 | 80.0 | 8.0 | 7238.0 | 17.3 | 1710.6 | 1321.0 | 72.5 |
| DL 40 | 26.2 | 162.1 | 80.1 | 8.0 | 7238.8 | 10.7 | 1719.5 | 771.7 | 72.4 |
| DL 43 | 28.2 | 162.1 | 80.3 | 8.0 | 7029.2 | 36.1 | 1652.2 | 2393.6 | 72.7 |
| DL 45 | 26.5 | 162.1 | 80.2 | 8.0 | 6330.2 | 35.3 | 1528.4 | 2776.7 | 72.7 |
| DL 47 | 27.8 | 163.3 | 80.5 | 8.0 | 5755.5 | 25.5 | 1447.8 | 1621.6 | 72.7 |
| DL 50 | 24.2 | 162.9 | 80.6 | 8.0 | 4700.3 | 17.6 | 1268.8 | 988.4 | 72.7 |
| DL 53 | 17.5 | 162.6 | 79.8 | 8.0 | 4515.6 | 2.3 | 1221.1 | 61.6 | 72.5 |
| DL 55 | 27.1 | 162.6 | 79.9 | 8.0 | 4280.4 | 2.3 | 1188.4 | 60.6 | 72.6 |
| DL 60 | 26.0 | 162.1 | 79.9 | 8.0 | 3653.2 | 1.9 | 1087.0 | 45.6 | 72.6 |
| DL 65 | 26.9 | 162.9 | 80.0 | 8.0 | 3930.3 | 1.6 | 1158.0 | 43.0 | 72.5 |
| DL 70 | 24.2 | 163.3 | 80.0 | 8.0 | 3553.3 | 1.3 | 1094.1 | 35.9 | 72.3 |
| DL 80 | 16.4 | 162.2 | 79.7 | 8.0 | 2895.6 | 0.3 | 1035.9 | 13.2 | 72.5 |
| DL 90 | 16.0 | 163.0 | 80.1 | 8.0 | 2904.1 | 0 | 1055.9 | 0 | 72.5 |

Table B.3: Results for 7-bit LED Set (10% noise) with Percentage Thresholds

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{DL}$ 0 | 67.8 | 154.1 | 75.1 | 8.0 | 5630.8 | 91.6 | 904.5 | 2176.4 | 72.6 |
| $\widehat{DL}$ 5 | 43.4 | 155.8 | 77.0 | 8.0 | 4835.7 | 64.2 | 802.7 | 3499.8 | 72.6 |
| $\widehat{DL}$ 10 | 38.2 | 157.2 | 77.3 | 8.0 | 4342.1 | 41.3 | 745.5 | 2643.6 | 72.6 |
| $\widehat{DL}$ 15 | 31.7 | 159.6 | 78.4 | 8.0 | 3984.3 | 38.3 | 702.9 | 2646.4 | 72.5 |
| $\widehat{DL}$ 20 | 28.2 | 157.3 | 77.8 | 8.0 | 3749.8 | 25.3 | 674.1 | 1772.4 | 72.5 |
| $\widehat{DL}$ 25 | 26.6 | 158.6 | 78.1 | 8.0 | 3312.9 | 16.9 | 610.7 | 1130.0 | 72.3 |
| $\widehat{DL}$ 30 | 22.6 | 158.0 | 78.0 | 8.0 | 3134.8 | 10.5 | 593.8 | 767.5 | 72.4 |
| $\widehat{DL}$ 35 | 23.3 | 159.8 | 78.8 | 8.0 | 2884.8 | 6.4 | 552.8 | 323.7 | 72.5 |
| $\widehat{DL}$ 40 | 23.2 | 159.8 | 79.0 | 8.0 | 2878.5 | 4.7 | 547.1 | 182.1 | 72.5 |
| $\widehat{DL}$ 43 | 24.3 | 160.3 | 79.1 | 8.0 | 2663.8 | 17.3 | 509.6 | 1260.9 | 72.6 |
| $\widehat{DL}$ 45 | 24.3 | 160.2 | 78.8 | 8.0 | 2448.0 | 12.7 | 479.0 | 764.2 | 72.6 |
| $\widehat{DL}$ 47 | 28.0 | 160.6 | 78.9 | 8.0 | 2193.9 | 10.2 | 437.2 | 514.8 | 72.6 |
| $\widehat{DL}$ 50 | 22.4 | 159.6 | 78.7 | 8.0 | 1833.1 | 7.7 | 387.2 | 368.1 | 72.6 |
| $\widehat{DL}$ 53 | 16.5 | 160.0 | 78.5 | 8.0 | 1940.7 | 2.1 | 404.2 | 64.1 | 72.5 |
| $\widehat{DL}$ 55 | 18.2 | 160.4 | 78.8 | 8.0 | 1867.5 | 2.1 | 393.1 | 62.4 | 72.6 |
| $\widehat{DL}$ 60 | 17.8 | 159.8 | 78.7 | 8.0 | 1591.5 | 1.4 | 350.8 | 36.6 | 72.6 |
| $\widehat{DL}$ 65 | 18.8 | 160.7 | 78.8 | 8.0 | 1668.7 | 1.3 | 371.0 | 39.4 | 72.4 |
| $\widehat{DL}$ 70 | 16.7 | 161.2 | 79.0 | 8.0 | 1745.2 | 1.2 | 385.2 | 38.4 | 72.3 |
| $\widehat{DL}$ 80 | 14.9 | 159.4 | 78.3 | 8.0 | 1313.8 | 0.3 | 335.0 | 13.2 | 72.5 |
| $\widehat{DL}$ 90 | 14.7 | 160.4 | 78.8 | 8.0 | 1292.3 | 0 | 339.7 | 0 | 72.5 |
| $\widehat{DL}$ 100 | 13.6 | 160.4 | 78.8 | 8.0 | 1292.3 | 0 | 339.7 | 0 | 72.5 |

Table B.4: Results for 7-bit LED Set (10% noise) with Percentage Thresholds

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| PJ 0 | 288.3 | 159.0 | 77.4 | 8.0 | 16758.8 | 204.2 | 3261.8 | 3219.8 | 72.5 |
| PJ 25 | 96.5 | 160.3 | 79.1 | 8.0 | 17958.7 | 13.7 | 2542.3 | 610.6 | 72.7 |
| PJ 50 | 83.4 | 163.4 | 80.1 | 8.0 | 18078.8 | 6.4 | 2307.4 | 369.4 | 72.6 |
| PJ 100 | 79.1 | 161.9 | 80.0 | 8.0 | 18128.4 | 3.1 | 2012.4 | 228.6 | 72.5 |
| PJ 150 | 75.0 | 161.3 | 79.7 | 8.0 | 18130.9 | 2.4 | 2002.0 | 192.1 | 72.7 |
| PJ 200 | 68.0 | 162.5 | 80.2 | 8.0 | 18147.9 | 1.3 | 1871.3 | 132.5 | 72.5 |
| PJ 300 | 63.2 | 161.8 | 80.0 | 8.0 | 18142.0 | 0.8 | 1766.2 | 98.1 | 72.5 |
| PJ 400 | 69.8 | 161.8 | 80.2 | 8.0 | 18141.1 | 0.8 | 1738.2 | 108.8 | 72.5 |

Table B.5: Results for 7-bit LED Set (10% noise) with Absolute Thresholds

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{PJ}$ 0 | 103.9 | 152.8 | 74.4 | 8.0 | 5411.2 | 127.8 | 890.9 | 1841.6 | 72.5 |
| $\widehat{PJ}$ 25 | 37.4 | 159.3 | 78.3 | 8.0 | 6130.2 | 4.1 | 659.4 | 251.3 | 72.7 |
| $\widehat{PJ}$ 50 | 34.3 | 158.8 | 78.0 | 8.0 | 6166.1 | 1.9 | 578.5 | 165.3 | 72.5 |
| $\widehat{PJ}$ 100 | 26.8 | 158.8 | 78.5 | 8.0 | 6172.1 | 0.9 | 549.8 | 101.9 | 72.5 |
| $\widehat{PJ}$ 150 | 31.4 | 159.7 | 79.0 | 8.0 | 6185.8 | 0.7 | 525.2 | 94.8 | 72.5 |
| $\widehat{PJ}$ 200 | 20.4 | 160.4 | 78.8 | 8.0 | 6194.9 | 0 | 521.0 | 0 | 72.5 |
| $\widehat{PJ}$ 300 | 20.1 | 160.4 | 78.8 | 8.0 | 6194.9 | 0 | 521.0 | 0 | 72.5 |
| $\widehat{PJ}$ 400 | 20.6 | 160.4 | 78.8 | 8.0 | 6194.9 | 0 | 521.0 | 0 | 72.5 |

Table B.6: Results for 7-bit LED Set (10% noise) with Absolute Thresholds

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| PJ 0 | 306.6 | 159.1 | 77.4 | 8.0 | 16717.7 | 217.1 | 3294.0 | 3330.9 | 72.6 |
| PJ 5 | 57.3 | 161.0 | 79.3 | 8.0 | 17721.7 | 27.9 | 2821.6 | 732.1 | 72.6 |
| PJ 10 | 47.0 | 162.8 | 80.0 | 8.0 | 17902.3 | 15.5 | 2663.6 | 450.8 | 72.6 |
| PJ 15 | 41.9 | 161.9 | 79.7 | 8.0 | 17919.9 | 10.4 | 2554.6 | 320.1 | 72.6 |
| PJ 20 | 40.7 | 162.7 | 80.1 | 8.0 | 17973.9 | 7.8 | 2468.0 | 260.3 | 72.5 |
| PJ 25 | 41.0 | 163.3 | 80.4 | 8.0 | 18045.8 | 6.4 | 2303.3 | 236.8 | 72.4 |
| PJ 30 | 38.5 | 162.6 | 80.2 | 8.0 | 18057.0 | 4.5 | 2221.9 | 176.9 | 72.6 |
| PJ 35 | 36.2 | 163.4 | 80.4 | 8.0 | 18055.5 | 3.6 | 2109.9 | 147.1 | 72.5 |
| PJ 40 | 33.5 | 163.8 | 80.4 | 8.0 | 18084.2 | 3.0 | 2056.7 | 115.8 | 72.5 |
| PJ 45 | 36.6 | 162.8 | 80.0 | 8.0 | 18101.4 | 2.1 | 1884.3 | 91.1 | 72.4 |
| PJ 50 | 31.9 | 162.3 | 79.9 | 8.0 | 18091.2 | 1.8 | 1810.5 | 69.9 | 72.6 |
| PJ 55 | 30.6 | 162.1 | 79.8 | 8.0 | 18121.1 | 1.3 | 1731.2 | 41.8 | 72.7 |
| PJ 60 | 31.0 | 162.6 | 80.0 | 8.0 | 18133.5 | 1.0 | 1768.3 | 28.9 | 72.4 |
| PJ 70 | 29.2 | 162.8 | 79.9 | 8.0 | 18112.8 | 0.9 | 1850.5 | 36.7 | 72.4 |
| PJ 80 | 28.7 | 162.7 | 80.0 | 8.0 | 18144.9 | 0.2 | 1714.3 | 7.3 | 72.5 |
| PJ 90 | 28.3 | 163.0 | 80.1 | 8.0 | 18157.5 | 0 | 1694.8 | 0 | 72.5 |
| PJ 100 | 29.7 | 163.0 | 80.1 | 8.0 | 18157.5 | 0 | 1694.8 | 0 | 72.5 |

Table B.7: Results for 7-bit LED Set (10% noise) with Percentage Thresholds

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{PJ}$ 0 | 100.3 | 152.8 | 74.4 | 8.0 | 5411.2 | 127.8 | 890.9 | 1841.6 | 72.5 |
| $\widehat{PJ}$ 5 | 31.7 | 156.8 | 77.2 | 8.0 | 6015.3 | 11.2 | 778.1 | 301.1 | 72.4 |
| $\widehat{PJ}$ 10 | 25.8 | 158.0 | 77.9 | 8.0 | 6046.4 | 5.9 | 716.0 | 149.0 | 72.3 |
| $\widehat{PJ}$ 15 | 27.4 | 160.4 | 78.7 | 8.0 | 6105.5 | 3.8 | 589.0 | 116.7 | 72.6 |
| $\widehat{PJ}$ 20 | 18.4 | 159.9 | 78.6 | 8.0 | 6176.3 | 2.5 | 570.5 | 39.4 | 72.5 |
| $\widehat{PJ}$ 25 | 19.4 | 159.9 | 78.8 | 8.0 | 6171.4 | 2.1 | 535.4 | 39.7 | 72.6 |
| $\widehat{PJ}$ 30 | 18.3 | 159.0 | 78.4 | 8.0 | 6161.2 | 1.5 | 566.1 | 27.4 | 72.5 |
| $\widehat{PJ}$ 35 | 19.1 | 160.2 | 78.8 | 8.0 | 6203.3 | 1.0 | 568.1 | 17.8 | 72.5 |
| $\widehat{PJ}$ 40 | 18.1 | 160.2 | 78.8 | 8.0 | 6214.8 | 1.0 | 563.2 | 20.3 | 72.5 |
| $\widehat{PJ}$ 45 | 19.8 | 159.7 | 78.4 | 8.0 | 6207.1 | 0.4 | 523.3 | 6.3 | 72.5 |
| $\widehat{PJ}$ 50 | 18.1 | 159.7 | 78.4 | 8.0 | 6206.4 | 0.4 | 522.1 | 7.1 | 72.5 |
| $\widehat{PJ}$ 60 | 18.7 | 159.9 | 78.6 | 8.0 | 6184.6 | 0.1 | 511.7 | 2.4 | 72.5 |
| $\widehat{PJ}$ 70 | 18.4 | 160.4 | 78.8 | 8.0 | 6194.9 | 0 | 521.0 | 0 | 72.5 |
| $\widehat{PJ}$ 80 | 19.2 | 160.4 | 78.8 | 8.0 | 6194.9 | 0 | 521.0 | 0 | 72.5 |
| $\widehat{PJ}$ 90 | 18.5 | 160.4 | 78.8 | 8.0 | 6194.9 | 0 | 521.0 | 0 | 72.5 |
| $\widehat{PJ}$ 100 | 18.3 | 160.4 | 78.8 | 8.0 | 6194.9 | 0 | 521.0 | 0 | 72.5 |

Table B.8: Results for 7-bit LED Set (10% noise) with Percentage Thresholds

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| QJ 1 | 21.0 | 158.6 | 78.5 | 8.0 | 1949.1 | 35.6 | 148.7 | 420.7 | 72.6 |
| QJ 1.25 | 34.1 | 157.0 | 77.7 | 8.0 | 3878.1 | 61.9 | 359.4 | 804.8 | 72.7 |
| QJ 1.5 | 58.9 | 156.6 | 77.4 | 8.0 | 7134.3 | 79.3 | 702.8 | 1392.1 | 72.7 |
| QJ 1.75 | 80.5 | 157.0 | 77.6 | 8.0 | 11063.3 | 94.9 | 1214.8 | 1865.9 | 72.6 |
| QJ 2 | 88.8 | 157.0 | 77.6 | 8.0 | 12828.5 | 107.3 | 1449.3 | 2071.7 | 72.6 |
| QJ 4 | 105.1 | 155.9 | 77.3 | 8.0 | 18122.9 | 148.9 | 2326.8 | 2527.2 | 72.7 |

Table B.9: Results for 7-bit LED Set (10% noise)

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| QJ-TDEC 1 | 97.6 | 156.7 | 77.5 | 8.0 | 14216.0 | 80.3 | 1552.4 | 2042.3 | 72.7 |
| QJ-TDEC 1.25 | 102.6 | 156.5 | 77.4 | 8.0 | 15594.2 | 106.1 | 1765.6 | 2246.6 | 72.7 |
| QJ-TDEC 1.5 | 104.5 | 156.5 | 77.4 | 8.0 | 16217.2 | 112.7 | 1886.1 | 2295.8 | 72.7 |
| QJ-TDEC 1.75 | 105.7 | 156.4 | 77.4 | 8.0 | 16927.9 | 122.7 | 2033.1 | 2364.0 | 72.7 |
| QJ-TDEC 2 | 109.4 | 156.4 | 77.4 | 8.0 | 17304.9 | 126.5 | 2111.5 | 2403.6 | 72.7 |
| QJ-TDEC 4 | 116.8 | 155.7 | 77.2 | 8.0 | 19995.2 | 158.4 | 2671.2 | 2600.5 | 72.7 |

Table B.10: Results for 7-bit LED Set (10% noise)

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{QJ}$ 1 | 20.8 | 159.7 | 78.7 | 8.0 | 1295.5 | 21.7 | 73.6 | 287.8 | 72.6 |
| $\widehat{QJ}$ 1.25 | 29.8 | 157.6 | 77.7 | 8.0 | 2089.9 | 35.2 | 135.9 | 496.1 | 72.7 |
| $\widehat{QJ}$ 1.50 | 36.1 | 158.4 | 78.1 | 8.0 | 2978.6 | 43.0 | 229.3 | 662.9 | 72.7 |
| $\widehat{QJ}$ 1.75 | 53.3 | 158.5 | 78.4 | 8.0 | 4275.0 | 52.7 | 346.3 | 1000.5 | 72.7 |
| $\widehat{QJ}$ 2 | 57.6 | 157.7 | 78.0 | 8.0 | 4869.9 | 59.5 | 403.7 | 1147.9 | 72.7 |
| $\widehat{QJ}$ 4 | 62.1 | 156.0 | 77.3 | 8.0 | 6552.5 | 81.4 | 621.3 | 1382.9 | 72.8 |

Table B.11: Results for 7-bit LED Set (10% noise)

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 9.6 | 158.0 | 79.5 | 8.0 | 236.0 | 0 | 0 | 0 | 72.7 |
| $\widehat{ID5}$ | 104.6 | 152.8 | 74.4 | 8.0 | 5411.2 | 127.8 | 890.9 | 1841.6 | 72.5 |
| J1 | 36.9 | 161.5 | 79.0 | 8.0 | 3761.6 | 98.5 | 637.8 | 919.6 | 72.4 |
| J2 | 13.8 | 161.4 | 79.5 | 8.0 | 462.1 | 7.4 | 11.1 | 46.1 | 72.5 |
| JTD | 36.0 | 160.7 | 78.5 | 8.0 | 3872.5 | 70.1 | 631.5 | 683.4 | 72.5 |

Table B.12: Results for 7-bit LED Set (10% noise)

### B.2.2 Soybean Disease

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| DL 0 | 1.5 | 9.3 | 7.0 | 3.3 | 1510.6 | 2.0 | 28.3 | 12.5 | 86.7 |
| DL 1 | 1.6 | 9.3 | 7.0 | 3.3 | 1510.6 | 2.0 | 28.3 | 12.5 | 86.7 |
| DL 2 | 1.6 | 9.3 | 7.0 | 3.3 | 1510.6 | 2.0 | 28.3 | 12.5 | 86.7 |
| DL 3 | 1.5 | 9.7 | 7.3 | 3.4 | 1465.9 | 1.8 | 24.6 | 15.1 | 84.0 |
| DL 4 | 1.7 | 9.3 | 7.0 | 3.3 | 1403.7 | 1.6 | 20.1 | 17.8 | 86.7 |
| DL 5 | 1.5 | 9.8 | 7.4 | 3.4 | 1304.4 | 1.2 | 17.8 | 16.1 | 84.7 |
| DL 6 | 1.6 | 13.1 | 8.9 | 3.5 | 1201.2 | 0.7 | 13.8 | 11.8 | 83.3 |
| DL 7 | 1.4 | 22.9 | 14.6 | 4.1 | 1126.1 | 0.1 | 13.3 | 2.3 | 69.3 |
| DL 8 | 1.1 | 22.9 | 14.6 | 4.1 | 1129.1 | 0.1 | 13.2 | 2.5 | 69.3 |
| DL 9 | 1.2 | 24.5 | 15.5 | 4.3 | 1122.0 | 0 | 13.2 | 0 | 66.7 |
| DL 10 | 1.0 | 24.5 | 15.5 | 4.3 | 1122.0 | 0 | 13.2 | 0 | 66.7 |

Table B.13: Results for Soybean Set - Absolute Thresholds

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{\text{DL}}$ 0 | 1.3 | 9.3 | 7.0 | 3.3 | 685.2 | 1.5 | 6.2 | 8.6 | 86.7 |
| $\widehat{\text{DL}}$ 1 | 1.4 | 9.3 | 7.0 | 3.3 | 685.2 | 1.5 | 6.2 | 8.6 | 86.7 |
| $\widehat{\text{DL}}$ 2 | 1.3 | 9.3 | 7.0 | 3.3 | 685.2 | 1.5 | 6.2 | 8.6 | 86.7 |
| $\widehat{\text{DL}}$ 3 | 1.3 | 11.4 | 8.3 | 3.6 | 690.8 | 1.0 | 4.8 | 5.5 | 84.7 |
| $\widehat{\text{DL}}$ 4 | 1.4 | 17.3 | 11.4 | 3.6 | 711.8 | 0.5 | 3.3 | 7.2 | 72.7 |
| $\widehat{\text{DL}}$ 5 | 1.2 | 22.9 | 14.6 | 4.1 | 683.3 | 0.1 | 2.2 | 2.3 | 69.3 |
| $\widehat{\text{DL}}$ 6 | 1.3 | 22.9 | 14.6 | 4.1 | 683.3 | 0.1 | 2.2 | 2.3 | 69.3 |
| $\widehat{\text{DL}}$ 7 | 1.3 | 22.9 | 14.6 | 4.1 | 683.3 | 0.1 | 2.2 | 2.3 | 69.3 |
| $\widehat{\text{DL}}$ 8 | 1.1 | 24.5 | 15.5 | 4.3 | 679.4 | 0 | 2.2 | 0 | 66.7 |
| $\widehat{\text{DL}}$ 9 | 1.1 | 24.5 | 15.5 | 4.3 | 679.4 | 0 | 2.2 | 0 | 66.7 |
| $\widehat{\text{DL}}$ 10 | 1.1 | 24.5 | 15.5 | 4.3 | 679.4 | 0 | 2.2 | 0 | 66.7 |

Table B.14: Results for Soybean Set - Absolute Thresholds

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| DL 0 | 2.9 | 9.3 | 7.0 | 3.3 | 1510.6 | 2.0 | 28.3 | 12.5 | 86.7 |
| DL 5 | 2.8 | 9.3 | 7.0 | 3.3 | 1510.6 | 2.0 | 28.3 | 12.5 | 86.7 |
| DL 10 | 2.6 | 9.7 | 7.3 | 3.4 | 1465.2 | 1.9 | 27.3 | 12.0 | 84.0 |
| DL 15 | 2.5 | 9.8 | 7.4 | 3.4 | 1405.2 | 1.6 | 24.9 | 11.9 | 86.0 |
| DL 20 | 2.3 | 12.6 | 9.3 | 3.8 | 1332.5 | 1.2 | 21.0 | 9.1 | 84.7 |
| DL 25 | 2.1 | 15.4 | 10.8 | 3.9 | 1257.1 | 0.8 | 18.7 | 6.4 | 82.0 |
| DL 30 | 2.0 | 21.5 | 13.9 | 4.0 | 1173.8 | 0.3 | 15.5 | 2.3 | 70.7 |
| DL 35 | 2.0 | 22.9 | 14.6 | 4.1 | 1146.8 | 0.2 | 13.4 | 2.4 | 69.3 |
| DL 40 | 2.1 | 22.9 | 14.6 | 4.1 | 1156.3 | 0.2 | 13.5 | 3.3 | 69.3 |
| DL 45 | 2.0 | 25.2 | 16.1 | 4.3 | 1145.2 | 0.1 | 13.4 | 0.4 | 66.7 |
| DL 50 | 2.0 | 25.2 | 16.1 | 4.3 | 1145.2 | 0.1 | 13.4 | 0.4 | 66.7 |
| DL 55 | 1.9 | 24.5 | 15.5 | 4.3 | 1122.0 | 0 | 13.2 | 0 | 66.7 |
| DL 60 | 1.9 | 24.5 | 15.5 | 4.3 | 1122.0 | 0 | 13.2 | 0 | 66.7 |
| DL 65 | 1.1 | 24.5 | 15.5 | 4.3 | 1122.0 | 0 | 13.2 | 0 | 66.7 |
| DL 70 | 1.1 | 24.5 | 15.5 | 4.3 | 1122.0 | 0 | 13.2 | 0 | 66.7 |

Table B.15: Results for Soybean Set - Percentage Thresholds

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{\text{DL}}$ 0 | 2.2 | 9.3 | 7.0 | 3.3 | 685.2 | 1.5 | 6.2 | 8.6 | 86.7 |
| $\widehat{\text{DL}}$ 5 | 2.2 | 9.3 | 7.0 | 3.3 | 685.2 | 1.5 | 6.2 | 8.6 | 86.7 |
| $\widehat{\text{DL}}$ 10 | 2.1 | 10.2 | 7.7 | 3.5 | 678.0 | 1.3 | 6.2 | 6.9 | 88.0 |
| $\widehat{\text{DL}}$ 15 | 1.9 | 13.7 | 9.8 | 3.8 | 670.3 | 0.9 | 5.8 | 4.7 | 79.3 |
| $\widehat{\text{DL}}$ 20 | 1.9 | 17.5 | 11.8 | 3.9 | 678.7 | 0.6 | 4.6 | 4.0 | 73.3 |
| $\widehat{\text{DL}}$ 25 | 1.9 | 17.5 | 11.8 | 3.9 | 678.5 | 0.6 | 4.4 | 4.2 | 73.3 |
| $\widehat{\text{DL}}$ 30 | 2.0 | 21.5 | 13.9 | 4.0 | 686.9 | 0.3 | 3.2 | 2.5 | 70.7 |
| $\widehat{\text{DL}}$ 35 | 2.0 | 22.9 | 14.6 | 4.1 | 699.5 | 0.2 | 2.7 | 2.4 | 69.3 |
| $\widehat{\text{DL}}$ 40 | 2.1 | 22.9 | 14.6 | 4.1 | 716.0 | 0.2 | 2.5 | 3.3 | 69.3 |
| $\widehat{\text{DL}}$ 45 | 2.0 | 25.2 | 16.1 | 4.3 | 705.3 | 0.1 | 2.4 | 0.4 | 66.7 |
| $\widehat{\text{DL}}$ 50 | 2.0 | 25.2 | 16.1 | 4.3 | 705.3 | 0.1 | 2.4 | 0 | 66.7 |
| $\widehat{\text{DL}}$ 55 | 1.9 | 24.5 | 15.5 | 4.3 | 679.4 | 0 | 2.2 | 0 | 66.7 |
| $\widehat{\text{DL}}$ 60 | 1.9 | 24.5 | 15.5 | 4.3 | 679.4 | 0 | 2.2 | 0 | 66.7 |
| $\widehat{\text{DL}}$ 70 | 1.8 | 24.5 | 15.5 | 4.3 | 679.4 | 0 | 2.2 | 0 | 66.7 |

Table B.16: Results for Soybean Set - Percentage Thresholds

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| PJ 0 | 1.6 | 7.7 | 5.7 | 3.0 | 1450.1 | 3.0 | 33.0 | 9.9 | 93.3 |
| PJ 1 | 1.6 | 9.3 | 7.0 | 3.3 | 1510.6 | 2.0 | 28.3 | 12.5 | 86.7 |
| PJ 2 | 1.5 | 9.7 | 7.3 | 3.4 | 1577.8 | 1.5 | 24.8 | 13.4 | 84.0 |
| PJ 3 | 1.7 | 9.3 | 7.0 | 3.3 | 1648.5 | 1.5 | 20.5 | 17.3 | 86.7 |
| PJ 4 | 1.8 | 9.8 | 7.4 | 3.4 | 1699.4 | 1.2 | 18.3 | 16.1 | 84.7 |
| PJ 5 | 1.7 | 13.1 | 8.9 | 3.5 | 1822.7 | 0.7 | 14.3 | 11.8 | 83.3 |
| PJ 6 | 1.5 | 22.9 | 14.6 | 4.1 | 1855.8 | 0.1 | 13.3 | 2.3 | 69.3 |
| PJ 7 | 1.7 | 22.9 | 14.6 | 4.1 | 1865.7 | 0.1 | 13.2 | 2.5 | 69.3 |
| PJ 8 | 1.5 | 24.5 | 15.5 | 4.3 | 1865.5 | 0 | 13.2 | 0 | 66.7 |
| PJ 9 | 1.5 | 24.5 | 15.5 | 4.3 | 1865.5 | 0 | 13.2 | 0 | 66.7 |

Table B.17: Results for Soybean Set - Absolute Thresholds

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{\text{PJ}}$ 0 | 1.1 | 7.9 | 5.9 | 3.0 | 500.1 | 2.3 | 7.4 | 6.3 | 94.0 |
| $\widehat{\text{PJ}}$ 1 | 1.4 | 9.3 | 7.0 | 3.3 | 685.2 | 1.5 | 6.2 | 8.6 | 86.7 |
| $\widehat{\text{PJ}}$ 2 | 1.4 | 14.9 | 10.0 | 3.6 | 780.5 | 0.7 | 3.2 | 3.4 | 81.3 |
| $\widehat{\text{PJ}}$ 3 | 1.3 | 20.7 | 13.3 | 3.9 | 858.2 | 0.2 | 2.2 | 4.6 | 70.0 |
| $\widehat{\text{PJ}}$ 4 | 1.3 | 24.5 | 15.5 | 4.3 | 861.0 | 0 | 2.2 | 0 | 66.7 |
| $\widehat{\text{PJ}}$ 5 | 1.3 | 24.5 | 15.5 | 4.3 | 861.0 | 0 | 2.2 | 0 | 66.7 |

Table B.18: Results for Soybean Set - Absolute Thresholds

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| PJ 0 | 2.9 | 7.7 | 5.7 | 3.0 | 1450.8 | 3.0 | 33.0 | 9.9 | 93.3 |
| PJ 5 | 2.7 | 9.7 | 7.3 | 3.4 | 1503.6 | 1.7 | 27.6 | 11.1 | 84.0 |
| PJ 10 | 2.8 | 9.8 | 7.4 | 3.4 | 1547.2 | 1.6 | 24.9 | 12.2 | 86.0 |
| PJ 15 | 2.6 | 12.0 | 8.8 | 3.7 | 1590.6 | 1.1 | 21.2 | 10.1 | 84.7 |
| PJ 20 | 2.6 | 19.2 | 12.7 | 4.0 | 1737.7 | 0.5 | 16.4 | 5.2 | 73.3 |
| PJ 25 | 2.7 | 22.9 | 14.6 | 4.1 | 1835.3 | 0.2 | 14.1 | 1.9 | 69.3 |
| PJ 30 | 2.7 | 22.9 | 14.6 | 4.1 | 1858.6 | 0.2 | 13.7 | 2.7 | 69.3 |
| PJ 35 | 2.6 | 24.5 | 15.5 | 4.3 | 1865.5 | 0 | 13.2 | 0 | 66.7 |
| PJ 40 | 2.6 | 24.5 | 15.5 | 4.3 | 1865.5 | 0 | 13.2 | 0 | 66.7 |

Table B.19: Results for Soybean Set - Percentage Thresholds

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{PJ}$ 0 | 1.9 | 7.9 | 5.9 | 3.0 | 500.1 | 2.3 | 7.4 | 6.3 | 94.0 |
| $\widehat{PJ}$ 5 | 2.1 | 10.2 | 7.7 | 3.5 | 692.0 | 1.3 | 6.2 | 6.9 | 88.0 |
| $\widehat{PJ}$ 10 | 2.0 | 21.2 | 13.7 | 4.0 | 779.4 | 0.3 | 3.3 | 1.7 | 69.3 |
| $\widehat{PJ}$ 15 | 2.0 | 21.2 | 13.7 | 4.0 | 813.3 | 0.3 | 3.2 | 2.0 | 69.3 |
| $\widehat{PJ}$ 20 | 2.1 | 22.9 | 14.6 | 4.1 | 853.9 | 0.2 | 2.7 | 1.9 | 69.3 |
| $\widehat{PJ}$ 25 | 2.2 | 22.9 | 14.6 | 4.1 | 877.5 | 0.2 | 2.7 | 2.4 | 69.3 |
| $\widehat{PJ}$ 30 | 2.1 | 24.5 | 15.5 | 4.3 | 861.0 | 0 | 2.2 | 0 | 66.7 |
| $\widehat{PJ}$ 35 | 2.1 | 24.5 | 15.5 | 4.3 | 861.0 | 0 | 2.2 | 0 | 66.7 |

Table B.20: Results for Soybean Set - Percentage Thresholds

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| QJ 1 | 1.5 | 22.5 | 14.3 | 4.1 | 318.4 | 0.2 | 0 | 2.5 | 68.0 |
| QJ 1.25 | 1.5 | 11.5 | 8.4 | 3.5 | 289.9 | 2.1 | 0.2 | 3.8 | 90.7 |
| QJ 1.50 | 1.5 | 11.5 | 8.4 | 3.5 | 289.9 | 2.1 | 0.2 | 3.8 | 90.7 |
| QJ 1.75 | 1.7 | 9.6 | 7.0 | 3.3 | 317.2 | 2.4 | 0.2 | 7.1 | 94.7 |
| QJ 2 | 1.7 | 8.7 | 6.4 | 3.2 | 331.4 | 2.6 | 0.2 | 9.0 | 92.7 |
| QJ 4 | 2.0 | 7.7 | 5.7 | 3.0 | 411.4 | 3.5 | 1.6 | 10.6 | 93.3 |

Table B.21: Results for Soybean Set

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| QJ-TDEC 1 | 1.8 | 10.0 | 7.4 | 3.6 | 337.5 | 1.1 | 0 | 10.1 | 87.3 |
| QJ-TDEC 1.25 | 1.7 | 8.0 | 6.0 | 3.0 | 322.8 | 2.7 | 0.1 | 9.1 | 91.3 |
| QJ-TDEC 1.50 | 1.8 | 8.0 | 6.0 | 3.0 | 353.9 | 3.0 | 0.2 | 10.0 | 91.3 |
| QJ-TDEC 1.75 | 1.9 | 8.0 | 6.0 | 3.0 | 386.1 | 3.2 | 1.0 | 10.5 | 91.3 |
| QJ-TDEC 2 | 1.9 | 7.7 | 5.7 | 3.0 | 400.0 | 3.3 | 1.1 | 11.3 | 93.3 |
| QJ-TDEC 4 | 2.3 | 7.7 | 5.7 | 3.0 | 603.6 | 3.9 | 6.0 | 12.4 | 93.3 |

Table B.22: Results for Soybean Set

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{QJ}$ 1 | 1.0 | 24.3 | 15.5 | 4.2 | 308.0 | 0.1 | 0 | 0.4 | 66.7 |
| $\widehat{QJ}$ 1.25 | 1.1 | 10.1 | 7.5 | 3.2 | 314.2 | 2.0 | 0 | 5.6 | 91.3 |
| $\widehat{QJ}$ 1.5 | 1.1 | 10.1 | 7.5 | 3.2 | 307.5 | 2.0 | 0 | 6.2 | 91.3 |
| $\widehat{QJ}$ 1.75 | 1.1 | 9.4 | 7.0 | 3.2 | 317.6 | 2.1 | 0 | 7.3 | 94.7 |
| $\widehat{QJ}$ 2 | 1.1 | 8.7 | 6.5 | 3.1 | 308.0 | 2.0 | 0 | 7.9 | 92.7 |
| $\widehat{QJ}$ 4 | 1.0 | 8.0 | 5.9 | 3.0 | 313.9 | 2.5 | 0.8 | 8.6 | 94.0 |

Table B.23: Results for Soybean Set

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 0.7 | 9.3 | 7.3 | 3.0 | 69.0 | 0 | 0 | 0 | 93.3 |
| $\widehat{ID5}$ | 1.2 | 7.9 | 5.9 | 3.0 | 500.1 | 2.3 | 7.4 | 6.3 | 94.0 |
| J1 | 1.1 | 18.3 | 13 | 4.0 | 398.3 | 2.0 | 1.8 | 3.7 | 76.0 |
| J2 | 1.1 | 20.9 | 14.3 | 4.1 | 411.7 | 1.9 | 1.7 | 3.3 | 70.7 |
| JTD | 1.3 | 17.4 | 12.4 | 4.0 | 391.9 | 1.9 | 1.8 | 3.9 | 78.0 |

Table B.24: Results for Soybean Set

### B.2.3 Chess

#### KRKPA7

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{DL}$ 0 | 1141.4 | 102.6 | 52.6 | 13.0 | 14084.8 | 29.9 | 229.9 | 873.6 | 97.6 |
| $\widehat{DL}$ 10 | 1212.9 | 138.1 | 71.1 | 14.8 | 12192.6 | 7.2 | 171.1 | 664.6 | 97.4 |
| $\widehat{DL}$ 20 | 960.4 | 145.3 | 75.0 | 15.0 | 9226.7 | 9.8 | 113.3 | 691.4 | 97.2 |
| $\widehat{DL}$ 30 | 652.7 | 189.3 | 97.5 | 14.4 | 9542.8 | 3.5 | 118.4 | 273.0 | 96.8 |
| $\widehat{DL}$ 40 | 724.5 | 187.4 | 96.5 | 14.3 | 8838.2 | 1.1 | 100.1 | 150.0 | 96.8 |
| $\widehat{DL}$ 50 | 288.0 | 210.6 | 107.6 | 14.0 | 8040.4 | 0.1 | 85.1 | 9.2 | 96.3 |
| $\widehat{DL}$ 100 | 259.1 | 211.9 | 108.1 | 14.0 | 8107.5 | 0 | 88.0 | 0 | 96.2 |
| $\widehat{PJ}$ 0 | 1084.7 | 87.0 | 45.0 | 12.8 | 14525.6 | 54.2 | 261.5 | 770.7 | 98.0 |
| $\widehat{PJ}$ 10 | 266.2 | 211.9 | 108.1 | 14.0 | 21415.2 | 0.0 | 113.9 | 0.0 | 96.2 |
| $\widehat{PJ}$ 20 | 266.5 | 211.9 | 108.1 | 14.0 | 21415.2 | 0.0 | 113.9 | 0.0 | 96.2 |

Table B.25: Results for KRKPA7 Chess Set

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| QJ 1 | 222.6 | 211.9 | 108.1 | 14.0 | 3016.5 | 0 | 0 | 0 | 96.2 |
| QJ 1.25 | 235.8 | 151.5 | 77.8 | 13.2 | 4755.6 | 16.5 | 18.7 | 70.4 | 97.4 |
| QJ 1.5 | 255.2 | 120.4 | 61.9 | 12.8 | 6875.0 | 20.2 | 57.0 | 119.2 | 97.7 |
| QJ 1.75 | 314.5 | 96.5 | 49.4 | 12.6 | 11712.4 | 27.6 | 144.7 | 192.1 | 96.9 |
| QJ 2 | 333.5 | 87.1 | 44.7 | 12.3 | 15294.9 | 30.6 | 229.0 | 237.1 | 96.7 |
| QJ 4 | 438.1 | 90.0 | 46.5 | 12.9 | 48078.9 | 55.5 | 1025.9 | 514.6 | 97.6 |
| $\widehat{\text{QJ}}$ 1 | 261.4 | 211.9 | 108.1 | 14.0 | 3016.5 | 0 | 0 | 0 | 96.2 |
| $\widehat{\text{QJ}}$ 1.25 | 264.0 | 179.7 | 91.8 | 13.3 | 3360.3 | 8.2 | 1.9 | 20.8 | 96.7 |
| $\widehat{\text{QJ}}$ 1.5 | 275.6 | 139.6 | 71.8 | 13.1 | 3626.0 | 8.8 | 3.0 | 48.7 | 97.4 |
| $\widehat{\text{QJ}}$ 1.75 | 280.6 | 133.7 | 68.9 | 13.3 | 4067.6 | 11.2 | 6.0 | 67.5 | 97.4 |
| $\widehat{\text{QJ}}$ 2 | 307.8 | 103.4 | 53.0 | 12.6 | 4561.8 | 12.3 | 11.2 | 75.2 | 97.0 |
| $\widehat{\text{QJ}}$ 4 | 393.7 | 88.8 | 45.8 | 12.1 | 9783.5 | 27.2 | 47.7 | 274.4 | 96.6 |

Table B.26: Results for KRKPA7 Chess Set

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 133.3 | 90.0 | 46.5 | 13.0 | 1278.7 | 0 | 0 | 0 | 98.1 |
| ID4 | 811.6 | 79.4 | 41.2 | 12.8 | 214635.0 | 70.3 | 5715.3 | 823.3 | 96.8 |
| $\widehat{\text{ID4}}$ | 542.0 | 66.4 | 34.7 | 12.3 | 15225.5 | 27.3 | 212.6 | 345.6 | 97.7 |
| ID5 | 2145.0 | 92.0 | 47.3 | 13.1 | 208063.0 | 122.1 | 5754.4 | 1609.1 | 98.1 |
| $\widehat{\text{ID5}}$ | 1149.4 | 87.0 | 45.0 | 12.8 | 14525.6 | 54.2 | 261.5 | 770.7 | 98.0 |
| ID5R | 2730.8 | 90.0 | 46.5 | 13.0 | 236318.0 | 178.4 | 6592.9 | 2053.1 | 98.1 |
| $\widehat{\text{ID5R}}$ | 2115.3 | 87.4 | 45.2 | 12.8 | 24957.1 | 85.8 | 555.1 | 956.5 | 98.1 |

Table B.27: Results for KRKPA7 Chess Set

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| J1 | 257.8 | 190.9 | 97.6 | 13.6 | 3256.9 | 7.9 | 2.0 | 18.3 | 96.8 |
| J2 | 250.4 | 193.8 | 99.1 | 13.6 | 3187.4 | 6.2 | 1.4 | 15.4 | 96.9 |
| JTD | 277.9 | 190.9 | 97.6 | 13.6 | 3227.7 | 7.1 | 2.1 | 15.9 | 96.8 |

Table B.28: Results for KRKPA7 Chess Set

### B.2.4 Knight Pins

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 8.9 | 16.9 | 11.4 | 5.3 | 79.1 | 0 | 0 | ? | 99.5 |
| $\widehat{\text{ID5}}$ | 15.8 | 16.1 | 10.8 | 5.2 | 311.3 | 2.0 | 10.3 | 6.5 | 99.5 |
| $\widehat{\text{QJ}}$ 1 | 18.7 | 22.3 | 14.3 | 6.1 | 110.1 | 0 | 0 | 0 | 99.3 |
| $\widehat{\text{QJ}}$ 2 | 18.7 | 21.0 | 13.7 | 5.6 | 124.7 | 0.6 | 0.1 | 1.6 | 99.4 |
| $\widehat{\text{QJ}}$ 4 | 19.0 | 19.5 | 12.8 | 5.5 | 146.6 | 1.3 | 0.2 | 4.5 | 99.4 |
| J1 | 13.6 | 21.0 | 13.8 | 5.5 | 153.3 | 1.6 | 0.2 | 4.2 | 99.5 |
| J2 | 13.5 | 22.9 | 15.0 | 5.6 | 144.2 | 1.0 | 0.2 | 2.2 | 99.4 |
| JTD | 15.2 | 23.9 | 15.7 | 5.6 | 164.4 | 1.6 | 0.2 | 3.5 | 99.3 |

Table B.29: Results for KRKK Chess Set

### B.2.5 Multiplexor

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{\text{DL}}$ 0 | 606.6 | 558.4 | 279.5 | 11.8 | 10004.9 | 101.9 | 481.4 | 10813.6 | 93.8 |
| $\widehat{\text{DL}}$ 10 | 207.3 | 652.8 | 326.9 | 11.8 | 8028.3 | 53.6 | 365.7 | 11068.4 | 92.4 |
| $\widehat{\text{DL}}$ 20 | 107.8 | 529.1 | 265.0 | 11.8 | 5493.3 | 18.7 | 234.8 | 2704.2 | 94.8 |
| $\widehat{\text{DL}}$ 30 | 106.1 | 530.8 | 265.9 | 11.7 | 4691.9 | 25.9 | 208.9 | 5553.5 | 95.2 |
| $\widehat{\text{DL}}$ 40 | 96.7 | 411.4 | 206.2 | 11.3 | 3614.9 | 5.0 | 148.7 | 478.7 | 97.0 |
| $\widehat{\text{DL}}$ 50 | 49.8 | 318.4 | 159.7 | 11.3 | 2364.5 | 1.4 | 91.3 | 94.2 | 98.0 |
| $\widehat{\text{DL}}$ 100 | 38.7 | 223.2 | 112.1 | 11.3 | 1436.9 | 0 | 40.4 | 0 | 98.7 |
| $\widehat{\text{PJ}}$ 0 | 1028.1 | 446.3 | 223.4 | 11.2 | 7344.2 | 179.9 | 411.3 | 9950.9 | 94.6 |
| $\widehat{\text{PJ}}$ 10 | 44.9 | 368.8 | 184.9 | 11.5 | 8354.6 | 2.6 | 237.3 | 72.6 | 96.9 |
| $\widehat{\text{PJ}}$ 20 | 41.1 | 320.4 | 160.7 | 11.5 | 7433.0 | 1.0 | 167.2 | 14.4 | 97.3 |
| $\widehat{\text{PJ}}$ 30 | 38.6 | 244.6 | 122.8 | 11.4 | 5771.4 | 0.4 | 107.5 | 11.7 | 98.7 |
| $\widehat{\text{PJ}}$ 40 | 40.1 | 223.2 | 112.1 | 11.3 | 5243.8 | 0 | 69.1 | 0 | 98.7 |
| $\widehat{\text{PJ}}$ 50 | 40.2 | 223.2 | 112.1 | 11.3 | 5243.8 | 0 | 69.1 | 0 | 98.7 |
| $\widehat{\text{PJ}}$ 100 | 40.1 | 223.2 | 112.1 | 11.3 | 5243.8 | 0 | 69.1 | 0 | 98.7 |

Table B.30: Results for 11-bit Multiplexor Set

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| QJ 1 | 31.5 | 223.2 | 112.1 | 11.3 | 584.2 | 0 | 0 | 0 | 98.7 |
| QJ 1.25 | 32.0 | 163.4 | 82.2 | 9.9 | 1351.0 | 36.1 | 37.7 | 98.1 | 99.1 |
| QJ 1.5 | 38.4 | 182.0 | 91.5 | 10.3 | 3084.7 | 69.4 | 157.3 | 254.2 | 98.9 |
| QJ 1.75 | 49.3 | 156.6 | 78.8 | 9.7 | 5473.2 | 88.8 | 328.1 | 503.4 | 99.5 |
| QJ 2 | 70.3 | 157.0 | 79.0 | 9.3 | 8916.2 | 118.7 | 563.9 | 854.6 | 99.3 |
| QJ 4 | 320.6 | 209.4 | 105.2 | 10.4 | 42976.4 | 335.5 | 2257.5 | 5568.8 | 98.9 |
| $\widehat{\text{QJ}}$ 1 | 32.6 | 223.2 | 112.1 | 11.3 | 584.2 | 0 | 0 | 0 | 98.7 |
| $\widehat{\text{QJ}}$ 1.25 | 32.7 | 164.6 | 82.8 | 10.1 | 754.1 | 13.6 | 5.1 | 32.7 | 99.2 |
| $\widehat{\text{QJ}}$ 1.5 | 33.6 | 180.2 | 90.6 | 10.2 | 948.1 | 18.2 | 7.8 | 56.3 | 98.9 |
| $\widehat{\text{QJ}}$ 1.75 | 34.3 | 146.2 | 73.6 | 9.8 | 1168.1 | 22.5 | 9.2 | 108.1 | 99.5 |
| $\widehat{\text{QJ}}$ 2 | 35.0 | 140.2 | 70.6 | 10.0 | 1339.8 | 24.3 | 14.0 | 135.3 | 99.6 |
| $\widehat{\text{QJ}}$ 4 | 73.1 | 216.6 | 108.8 | 10.2 | 7377.2 | 91.7 | 141.9 | 1238.0 | 99.0 |

Table B.31: Results for 11-bit Multiplexor Set

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 34.6 | 218.0 | 109.5 | 10.3 | 618.9 | 0.0 | 0.0 | 0.0 | 99.0 |
| $\widehat{\text{ID5}}$ | 1090.8 | 446.3 | 223.4 | 11.2 | 7344.2 | 179.9 | 411.3 | 9950.9 | 94.6 |
| J1 | 37.5 | 313.8 | 157.4 | 10.8 | 1520.9 | 43.8 | 19.1 | 117.5 | 97.7 |
| J2 | 38.2 | 285.2 | 143.1 | 11.6 | 1229.9 | 33.6 | 7.8 | 83.1 | 98.0 |
| JTD | 42.9 | 316.4 | 158.7 | 10.9 | 1442.3 | 36.1 | 17.3 | 105.8 | 97.7 |

Table B.32: Results for 11-bit Multiplexor Set

### B.2.6 Parity

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{\text{DL}}$ 0 | 1975.2 | 761.3 | 381.0 | 10.3 | 23579.5 | 285.8 | 1530.6 | 69721.8 | 82.3 |
| $\widehat{\text{DL}}$ 5 | 516.8 | 1619.9 | 810.4 | 12.0 | 22672.0 | 165.1 | 1179.6 | 82141.7 | 62.1 |
| $\widehat{\text{DL}}$ 10 | 294.3 | 1588.2 | 794.6 | 12.0 | 17762.5 | 91.1 | 928.1 | 55766.0 | 61.5 |
| $\widehat{\text{DL}}$ 20 | 175.9 | 1540.0 | 770.4 | 12.0 | 13377.5 | 66.5 | 745.1 | 49790.0 | 60.0 |
| $\widehat{\text{DL}}$ 30 | 158.7 | 1716.1 | 858.5 | 12.0 | 12280.4 | 36.7 | 738.7 | 18484.6 | 54.0 |
| $\widehat{\text{DL}}$ 40 | 137.7 | 1835.8 | 918.4 | 12.0 | 11074.0 | 36.4 | 713.4 | 30701.2 | 46.6 |
| $\widehat{\text{DL}}$ 50 | 96.1 | 1942.3 | 971.6 | 12.0 | 9531.3 | 2.8 | 611.1 | 452.6 | 41.6 |
| $\widehat{\text{DL}}$ 100 | 59.3 | 2075.6 | 1038.3 | 12.0 | 7939.8 | 0 | 526.8 | 0 | 34.4 |

Table B.33: Results for Parity Set

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{PJ}$ 10 | 158.9 | 1631.3 | 816.1 | 12.0 | 40661.0 | 6.6 | 1247.7 | 1179.3 | 55.4 |
| $\widehat{PJ}$ 20 | 111.4 | 1858.1 | 929.5 | 12.0 | 46653.3 | 2.8 | 1348.1 | 614.0 | 43.1 |
| $\widehat{PJ}$ 30 | 73.2 | 1953.8 | 977.4 | 12.0 | 49246.0 | 0.5 | 1275.3 | 95.2 | 39.9 |
| $\widehat{PJ}$ 40 | 76.6 | 2075.6 | 1038.3 | 12.0 | 52065.2 | 0 | 1440.3 | 0 | 34.4 |
| $\widehat{PJ}$ 50 | 76.2 | 2075.6 | 1038.3 | 12.0 | 52065.2 | 0 | 1440.3 | 0 | 34.4 |
| $\widehat{PJ}$ 100 | 76.3 | 2075.6 | 1038.3 | 12.0 | 52065.2 | 0 | 1440.3 | 0 | 34.4 |

Table B.34: Results for Parity Set

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| QJ 1 | 46.3 | 2075.6 | 1038.3 | 12.0 | 2815.9 | 0 | 0 | 0 | 34.4 |
| QJ 1.25 | 65.3 | 1875.9 | 938.2 | 12.0 | 8900.8 | 361.1 | 268.0 | 1534.7 | 46.5 |
| QJ 1.5 | 95.3 | 1762.0 | 881.1 | 12.0 | 14676.7 | 466.0 | 354.0 | 3507.8 | 49.5 |
| QJ 1.75 | 171.3 | 1560.8 | 780.6 | 12.0 | 27831.1 | 628.4 | 726.3 | 7900.5 | 56.0 |
| QJ 2 | 263.6 | 1512.0 | 756.2 | 12.0 | 41852.8 | 739.2 | 1140.2 | 12260.0 | 55.7 |
| QJ 4 | 1395.1 | 1359.2 | 679.8 | 12.0 | 174261.0 | 1477.3 | 4396.0 | 53150.9 | 59.8 |
| $\widehat{QJ}$ 1 | 48.1 | 2075.6 | 1038.3 | 12.0 | 2815.9 | 0 | 0 | 0 | 34.4 |
| $\widehat{QJ}$ 1.25 | 54.8 | 1534.7 | 767.8 | 12.0 | 5461.5 | 183.6 | 113.9 | 768.1 | 56.4 |
| $\widehat{QJ}$ 1.5 | 64.9 | 1211.7 | 606.2 | 12.0 | 7433.8 | 208.7 | 121.2 | 1659.4 | 67.3 |
| $\widehat{QJ}$ 1.75 | 87.0 | 945.9 | 473.3 | 12.0 | 11383.4 | 253.0 | 197.8 | 3110.1 | 75.5 |
| $\widehat{QJ}$ 2 | 114.7 | 805.6 | 403.2 | 11.7 | 15848.9 | 278.6 | 283.1 | 4715.4 | 80.4 |
| $\widehat{QJ}$ 4 | 443.3 | 648.9 | 324.7 | 11.3 | 58149.5 | 553.3 | 1068.5 | 18140.3 | 83.3 |

Table B.35: Results for Parity Set

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 38.7 | 1243.4 | 622.2 | 12.0 | 1837.3 | 0 | 0 | 0 | 62.9 |
| $\widehat{ID5}$ | 4956.9 | 1234.3 | 616.5 | 11.3 | 21842.8 | 557.9 | 1473.8 | 73494.5 | 64.7 |
| J1 | 313.3 | 592.8 | 296.8 | 12.0 | 8182.4 | 457.4 | 396.7 | 7795.3 | 90.5 |
| J2 | 111.2 | 1005.2 | 503.0 | 12.0 | 6359.6 | 379.7 | 294.0 | 2980.8 | 76.2 |
| JTD | 558.1 | 624.6 | 312.2 | 12.0 | 7996.2 | 274.4 | 474.5 | 10312.3 | 86.4 |

Table B.36: Results for Parity Set

### B.2.7 Balloons

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 0.03 | 5.0 | 3.0 | 3.0 | 7.0 | 0 | 0 | 0 | 100.0 |
| INC ID3 | 0.17 | 5.0 | 3.0 | 3.0 | 60.4 | 14.0 | 0 | ? | 100.0 |
| ID4 | 0.06 | 4.6 | 2.8 | 2.8 | 53.2 | 1.6 | 8.4 | 5.0 | 95.0 |
| $\widehat{\text{ID4}}$ | 0.06 | 4.4 | 2.7 | 2.7 | 26.1 | 1.6 | 1.1 | 4.8 | 90.0 |
| ID5 | 0.12 | 5.0 | 3.0 | 3.0 | 57.9 | 2.3 | 9.7 | 7.5 | 100.0 |
| $\widehat{\text{ID5}}$ | 0.10 | 5.0 | 3.0 | 3.0 | 26.8 | 2.1 | 1.4 | 6.1 | 100.0 |
| ID5R | 0.18 | 5.0 | 3.0 | 3.0 | 67.1 | 2.5 | 12.4 | 8.4 | 100.0 |
| $\widehat{\text{ID5R}}$ | 0.17 | 5.0 | 3.0 | 3.0 | 31.6 | 2.3 | 2.9 | 7.6 | 100.0 |

Table B.37: Results for Balloons Set 1

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 0.03 | 4.8 | 2.9 | 2.9 | 6.7 | 0 | 0 | 0 | 93.3 |
| INC ID3 | 0.18 | 4.8 | 2.9 | 2.9 | 60.1 | 14.0 | 0 | ? | 93.3 |
| ID4 | 0.05 | 3.6 | 2.3 | 2.3 | 52.9 | 1.4 | 9.0 | 3.8 | 83.3 |
| $\widehat{\text{ID4}}$ | 0.04 | 4.8 | 2.9 | 2.9 | 15.2 | 0.6 | 0.8 | 0.9 | 93.3 |
| ID5 | 0.1 | 4.8 | 2.9 | 2.9 | 54.2 | 1.7 | 9.8 | 4.5 | 93.3 |
| $\widehat{\text{ID5}}$ | 0.06 | 4.8 | 2.9 | 2.9 | 16.0 | 0.7 | 0.8 | 1.0 | 93.3 |
| ID5R | 0.08 | 4.8 | 2.9 | 2.9 | 64.2 | 1.7 | 12.8 | 4.5 | 93.3 |
| $\widehat{\text{ID5R}}$ | 0.08 | 4.8 | 2.9 | 2.9 | 16.0 | 0.7 | 0.8 | 1.0 | 93.3 |

Table B.38: Results for Balloons Set 2

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 0.03 | 5.0 | 3.0 | 3.0 | 7.0 | 0 | 0 | 0 | 100.0 |
| INC ID3 | 0.18 | 5.0 | 3.0 | 3.0 | 77.3 | 14 | 0 | ? | 100.0 |
| ID4 | 0.06 | 5.0 | 3.0 | 3.0 | 66.5 | 1.7 | 11.8 | 4.4 | 100.0 |
| $\widehat{\text{ID4}}$ | 0.04 | 5.0 | 3.0 | 3.0 | 14.3 | 0.8 | 0.1 | 1.3 | 100.0 |
| ID5 | 0.09 | 5.0 | 3.0 | 3.0 | 70.9 | 2.0 | 14.1 | 5.0 | 100.0 |
| $\widehat{\text{ID5}}$ | 0.05 | 5.0 | 3.0 | 3.0 | 16.7 | 1.1 | 0.1 | 1.9 | 100.0 |
| ID5R | 0.09 | 5.0 | 3.0 | 3.0 | 79.9 | 2.0 | 16.8 | 5.0 | 100.0 |
| $\widehat{\text{ID5R}}$ | 0.08 | 5.0 | 3.0 | 3.0 | 16.7 | 1.1 | 0.1 | 1.9 | 100.0 |

Table B.39: Results for Balloons Set 3

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 0.03 | 9.4 | 5.2 | 4.3 | 11.6 | 0 | 0 | 0 | 58.0 |
| INC ID3 | 0.17 | 9.4 | 5.2 | 4.3 | 85.1 | 11.0 | 0 | ? | 58.0 |
| ID4 | 0.08 | 9.0 | 5.0 | 4.3 | 63.0 | 2.7 | 6.5 | 10.0 | 60.0 |
| $\widehat{\text{ID4}}$ | 0.07 | 8.6 | 4.8 | 4.2 | 37.6 | 2.2 | 1.5 | 7.0 | 60.0 |
| ID5 | 0.19 | 9.8 | 5.4 | 4.3 | 70.9 | 4.6 | 7.3 | 17.5 | 54.0 |
| $\widehat{\text{ID5}}$ | 0.11 | 9.2 | 5.1 | 4.1 | 43.8 | 3.0 | 1.7 | 8.0 | 56.0 |
| ID5R | 0.17 | 9.4 | 5.2 | 4.3 | 98.4 | 5.8 | 16.2 | 23.5 | 58.0 |
| $\widehat{\text{ID5R}}$ | 0.15 | 9.2 | 5.1 | 4.1 | 48.1 | 3.3 | 3.1 | 8.7 | 60.0 |

Table B.40: Results for Balloons Set 4

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| J1 | 0.07 | 8.6 | 4.8 | 3.7 | 20.6 | 1.2 | 0.8 | 2.8 | 95.0 |
| J2 | 0.07 | 9.4 | 5.2 | 3.9 | 18.4 | 0.9 | 0.4 | 2.0 | 90.0 |
| JTD | 0.08 | 8.6 | 4.8 | 3.7 | 21.2 | 1.2 | 0.7 | 2.9 | 95.0 |

Table B.41: Results for Balloons Set 1

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| J1 | 0.05 | 4.8 | 2.9 | 2.9 | 9.1 | 0.3 | 0 | 0.6 | 93.3 |
| J2 | 0.04 | 4.8 | 2.9 | 2.9 | 10.7 | 0.4 | 0 | 0.6 | 93.3 |
| JTD | 0.04 | 4.8 | 2.9 | 2.9 | 9.1 | 0.3 | 0 | 0.6 | 93.3 |

Table B.42: Results for Balloons Set 2

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| J1 | 0.05 | 5.6 | 3.3 | 3.1 | 15.0 | 1.0 | 0.1 | 2.7 | 100.0 |
| J2 | 0.05 | 5.6 | 3.3 | 3.1 | 14.9 | 1.0 | 0.2 | 3.0 | 100.0 |
| JTD | 0.06 | 5.6 | 3.3 | 3.1 | 14.7 | 0.9 | 0.1 | 2.5 | 100.0 |

Table B.43: Results for Balloons Set 3

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| J1 | 0.12 | 10.4 | 5.7 | 4.3 | 46.3 | 3.4 | 2.8 | 11.2 | 58.0 |
| J2 | 0.08 | 10.0 | 5.5 | 4.2 | 30.2 | 2.3 | 1.0 | 6.4 | 64.0 |
| JTD | 0.11 | 10.0 | 5.5 | 4.2 | 47.3 | 3.2 | 2.4 | 9.9 | 56.0 |

Table B.44: Results for Balloons Set 4

### B.2.8 Lung Cancer

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 1.3 | 12.9 | 8.7 | 3.9 | 226.6 | 0 | 0 | 0 | 55.0 |
| INC ID3 | 11.3 | 12.9 | 8.7 | 3.9 | 2499.5 | 19.0 | 0 | ? | 55.0 |
| ID4 | 3.1 | 12.3 | 8.1 | 3.9 | 1737.1 | 5.9 | 10.7 | 22.8 | 55.0 |
| $\widehat{\text{ID4}}$ | 2.8 | 10.8 | 7.2 | 3.9 | 1263.1 | 5.9 | 2.8 | 21.5 | 55.0 |
| ID5 | 4.5 | 12.3 | 8.1 | 3.9 | 2454.4 | 9.0 | 15.2 | 37.6 | 55.0 |
| $\widehat{\text{ID5}}$ | 4.1 | 12.4 | 8.2 | 3.9 | 1806.3 | 8.5 | 4.2 | 34.2 | 56.3 |
| ID5R | 5.4 | 12.3 | 8.1 | 3.9 | 2638.5 | 9.4 | 17.7 | 38.5 | 55.0 |
| $\widehat{\text{ID5R}}$ | 5.7 | 12.4 | 8.2 | 3.9 | 1914.4 | 8.7 | 5.7 | 34.6 | 56.3 |

Table B.45: Results for Lung Cancer Set

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{QJ}$ 1 | 1.3 | 23.1 | 14.2 | 5.2 | 591.6 | 0.6 | 0 | 3.0 | 32.5 |
| $\widehat{QJ}$ 1.25 | 1.9 | 20.0 | 12.6 | 4.8 | 930.8 | 3.7 | 0.4 | 10.1 | 25.0 |
| $\widehat{QJ}$ 1.50 | 2.0 | 20.2 | 12.6 | 4.9 | 957.3 | 3.7 | 0.4 | 10.7 | 26.3 |
| $\widehat{QJ}$ 1.75 | 2.1 | 18.4 | 11.7 | 4.7 | 1012.0 | 4.1 | 0.4 | 14.0 | 30.0 |
| $\widehat{QJ}$ 2 | 2.2 | 18.0 | 11.5 | 4.7 | 1038.6 | 4.2 | 0.4 | 15.1 | 31.3 |
| $\widehat{QJ}$ 4 | 3.9 | 13.4 | 8.7 | 4.2 | 1657.8 | 7.9 | 0.9 | 35.6 | 48.8 |

Table B.46: Results for Lung Cancer Set

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| J1 | 2.7 | 17.6 | 11.3 | 4.7 | 1051.0 | 5.3 | 0.6 | 18.3 | 32.5 |
| J2 | 2.2 | 18.4 | 11.5 | 4.8 | 1005.7 | 4.3 | 0.4 | 14.2 | 35.0 |
| JTD | 2.4 | 18.4 | 11.6 | 4.8 | 1002.6 | 4.3 | 0.5 | 13.1 | 33.8 |

Table B.47: Results for Lung Cancer Set

### B.2.9 Cardio-Vascular

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{QJ}$ 1 | 11.0 | 217.9 | 83.0 | 27.0 | 2963.7 | 3.3 | 0.5 | 35.5 | 64.0 |
| $\widehat{QJ}$ 1.25 | 14.5 | 208.4 | 79.3 | 27.0 | 4557.6 | 13.9 | 6.6 | 120.8 | 65.4 |
| $\widehat{QJ}$ 1.50 | 14.7 | 200.9 | 78.0 | 27.0 | 4832.0 | 16.5 | 10.0 | 137.9 | 65.7 |
| $\widehat{QJ}$ 1.75 | 17.7 | 195.4 | 76.3 | 27.0 | 5880.5 | 19.4 | 15.1 | 192.0 | 65.0 |
| $\widehat{QJ}$ 2 | 20.0 | 198.6 | 75.1 | 27.0 | 6763.7 | 22.5 | 19.0 | 247.2 | 65.3 |
| $\widehat{QJ}$ 4 | 28.0 | 195.9 | 72.5 | 27.0 | 10952.2 | 46.2 | 49.3 | 460.9 | 65.8 |

Table B.48: Results for Cardio-Vascular Set

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 10.1 | 238.4 | 122.2 | 27.0 | 1922.3 | 0 | 0 | 0 | 64.8 |
| $\widehat{ID5}$ | 90.8 | 149.7 | 64.6 | 22.4 | 10669.8 | 53.8 | 145.0 | 504.9 | 66.5 |
| J1 | 64.1 | 153.4 | 68.9 | 21.7 | 9933.8 | 62.5 | 129.7 | 733.9 | 66.4 |
| J2 | 31.0 | 202.6 | 82.0 | 27.0 | 5076.2 | 30.2 | 32.4 | 306.5 | 63.7 |
| JTD | 31.7 | 154.1 | 65.3 | 27.0 | 5473.9 | 22.8 | 64.6 | 232.8 | 66.1 |

Table B.49: Results for Cardio-Vascular Set

### B.2.10 Breast-Cancer

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{QJ}$ 1 | 2.2 | 179.9 | 113.0 | 8.9 | 404.2 | 0 | 0 | 0 | 56.5 |
| $\widehat{QJ}$ 1.25 | 2.6 | 165.7 | 109.5 | 9.0 | 547.8 | 8.3 | 7.4 | 22.8 | 57.0 |
| $\widehat{QJ}$ 1.50 | 2.8 | 163.8 | 108.6 | 9.0 | 580.0 | 9.2 | 8.0 | 29.3 | 57.3 |
| $\widehat{QJ}$ 1.75 | 2.9 | 162.1 | 107.1 | 9.0 | 652.8 | 9.8 | 9.2 | 37.9 | 57.8 |
| $\widehat{QJ}$ 2 | 3.0 | 161.6 | 106.4 | 9.0 | 658.2 | 10.3 | 10.8 | 44.1 | 57.7 |
| $\widehat{QJ}$ 4 | 7.0 | 151.7 | 102.3 | 8.9 | 1667.5 | 25.9 | 35.8 | 236.0 | 58.1 |

Table B.50: Results for Breast Cancer Set

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 3.1 | 237.4 | 189.3 | 8.9 | 289.7 | 0 | 0 | 0 | 58.5 |
| $\widehat{ID5}$ | 17.3 | 149.7 | 102.2 | 8.5 | 2586.0 | 48.1 | 155.4 | 558.7 | 56.7 |
| J1 | 5.2 | 152.4 | 101.9 | 8.1 | 1178.2 | 26.6 | 61.5 | 145.1 | 57.7 |
| J2 | 2.7 | 173.2 | 109.7 | 8.9 | 547.3 | 6.5 | 9.3 | 19.6 | 55.9 |
| JTD | 4.8 | 147.0 | 98.2 | 8.2 | 959.7 | 17.4 | 45.5 | 93.0 | 57.1 |

Table B.51: Results for Breast Cancer Set

### B.2.11 Post-Op. Care

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 0.7 | 69.1 | 43.5 | 8.2 | 116.8 | 0 | 0 | 0 | 54.6 |
| $\widehat{ID5}$ | 5.5 | 53.8 | 29.8 | 8.1 | 796.0 | 21.4 | 43.2 | 202.7 | 52.9 |
| $\widehat{QJ}$ 2 | 1.2 | 62.1 | 34.1 | 8.2 | 337.5 | 9.1 | 5.1 | 39.8 | 49.3 |
| J1 | 4.3 | 53.6 | 30.0 | 8.3 | 633.6 | 22.4 | 34.5 | 181.5 | 53.9 |
| J2 | 1.8 | 61.9 | 33.9 | 8.3 | 367.0 | 13.5 | 10.8 | 63.7 | 55.7 |
| JTD | 2.6 | 47.8 | 26.7 | 7.8 | 408.3 | 9.8 | 21.2 | 86.2 | 55.4 |

Table B.52: Results for Post-Operative Care Set

### B.2.12 Noughts and Crosses

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 8.7 | 266.2 | 177.8 | 8.0 | 442.7 | 0 | 0 | 0 | 83.8 |
| $\widehat{\text{ID5}}$ | 63.9 | 260.2 | 164.3 | 8.2 | 5707.6 | 124.4 | 398.9 | 1588.3 | 83.8 |
| $\widehat{\text{QJ}}$ 2 | 11.7 | 299.2 | 188.0 | 8.1 | 1538.2 | 34.1 | 21.9 | 190.43 | 77.8 |
| J1 | 12.1 | 358.2 | 224.6 | 8.0 | 1385.5 | 48.3 | 25.3 | 141.5 | 74.8 |
| J2 | 11.6 | 388.6 | 241.6 | 8.2 | 1213.8 | 34.9 | 17.8 | 93.8 | 75.3 |
| JTD | 12.6 | 352.2 | 221.1 | 8.0 | 1305.4 | 40.9 | 22.1 | 126.2 | 74.4 |

Table B.53: Results for Noughts and Crosses Set

### B.2.13 Lenses

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 0.1 | 20.5 | 11.4 | 5.0 | 26.7 | 0 | 0 | 0 | 14.3 |
| $\widehat{\text{ID5}}$ | 0.4 | 18.0 | 9.1 | 4.9 | 116.4 | 7.9 | 14.2 | 36.9 | 14.3 |
| $\widehat{\text{QJ}}$ 2 | 0.2 | 18.3 | 9.3 | 5.0 | 82.9 | 3.9 | 5.5 | 16.8 | 14.3 |
| J1 | 0.4 | 18.8 | 9.3 | 5.0 | 138.0 | 9.6 | 18.8 | 44.9 | 14.3 |
| J2 | 0.2 | 18.3 | 9.2 | 4.9 | 79.6 | 4.7 | 7.1 | 18.7 | 14.3 |
| JTD | 0.3 | 17.5 | 8.9 | 4.8 | 104.0 | 6.4 | 11.7 | 28.8 | 14.3 |

Table B.54: Results for Lenses Set

## B.2.14 Lymphography

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 2.0 | 66.3 | 50.7 | 6.2 | 249.6 | 0 | 0 | 0 | 71.3 |
| $\widehat{\text{ID5}}$ | 9.8 | 53.9 | 37.0 | 5.9 | 1944.2 | 21.4 | 42 | 169.2 | 67.5 |
| $\widehat{\text{QJ}}$ 2 | 2.5 | 60.8 | 39.6 | 6.7 | 728.4 | 6.5 | 2.3 | 24.9 | 67.3 |
| J1 | 2.5 | 59.0 | 37.4 | 6.9 | 702.3 | 9.4 | 2.8 | 22.8 | 66.5 |
| J2 | 2.3 | 60.9 | 37.6 | 6.8 | 664.3 | 8.1 | 2.1 | 18.4 | 68.1 |
| JTD | 2.8 | 59.2 | 37.7 | 6.8 | 708.5 | 8.8 | 3.2 | 22.3 | 66.3 |

Table B.55: Results for Lymphography Set

## B.2.15 Monks Problems

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 0.7 | 94.0 | 62.0 | 7.0 | 100.0 | 0 | 0 | 0 | 78.0 |
| ID4 | 1.7 | 16.0 | 11.0 | 4.0 | 1370.0 | 9.0 | 211.0 | 58.0 | 80.6 |
| $\widehat{\text{ID4}}$ | 1.2 | 28.0 | 19.0 | 4.0 | 384.0 | 6.0 | 38.0 | 35.0 | 88.9 |
| ID5 | 3.2 | 84.0 | 51.0 | 7.0 | 1432.0 | 14.0 | 215.0 | 78.0 | 75.2 |
| $\widehat{\text{ID5}}$ | 1.9 | 65.0 | 40.0 | 7.0 | 369.0 | 7.0 | 40.0 | 36.0 | 85.4 |
| ID5R | 3.9 | 77.0 | 47.0 | 7.0 | 1736.0 | 25.0 | 291.0 | 169.0 | 78.0 |
| $\widehat{\text{ID5R}}$ | 2.0 | 63.0 | 39.0 | 7.0 | 563.0 | 13.0 | 90.0 | 76.0 | 86.3 |
| $\widehat{\text{DL}}$ 10 | 1.1 | 40.0 | 27.0 | 4.0 | 358.0 | 2.0 | 46.0 | 30.0 | 97.2 |
| $\widehat{\text{PJ}}$ 10 | 0.9 | 61.0 | 41.0 | 6.0 | 548.0 | 0 | 36.0 | 0 | 88.9 |
| $\widehat{\text{QJ}}$ 2 | 0.6 | 43.0 | 30.0 | 5.0 | 66.0 | 1.0 | 0 | 1.0 | 95.4 |
| J1 | 0.7 | 59.0 | 40.0 | 5.0 | 100.0 | 1.0 | 4.0 | 1.0 | 89.8 |
| J2 | 0.8 | 61.0 | 41.0 | 6.0 | 100.0 | 0.0 | 6.0 | 0 | 88.9 |
| JTD | 0.7 | 59.0 | 40.0 | 5.0 | 100.0 | 1.0 | 4.0 | 1.0 | 89.8 |

Table B.56: Results for Monks Set No. 1

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 1.2 | 173.0 | 109.0 | 7.0 | 168.0 | 0 | 0 | 0 | 65.0 |
| ID4 | 11.5 | 159.0 | 95.0 | 7.0 | 3553.0 | 46.0 | 282.0 | 950.0 | 65.0 |
| $\widehat{\text{ID4}}$ | 7.7 | 159.0 | 95.0 | 7.0 | 2172.0 | 35.0 | 134.0 | 577.0 | 65.0 |
| ID5 | 33.0 | 189.0 | 111.0 | 7.0 | 2249.0 | 86.0 | 279.0 | 1957.0 | 66.2 |
| $\widehat{\text{ID5}}$ | 24.7 | 185.0 | 109.0 | 7.0 | 1076.0 | 55.0 | 101.0 | 1559.0 | 67.4 |
| ID5R | 57.3 | 159.0 | 95.0 | 7.0 | 9010.0 | 346.0 | 2262.0 | 4279.0 | 65.0 |
| $\widehat{\text{ID5R}}$ | 27.2 | 174.0 | 104.0 | 7.0 | 4863.0 | 204.0 | 1112.0 | 2242.0 | 64.0 |
| $\widehat{\text{DL}}$ 10 | 5.9 | 187.0 | 108.0 | 7.0 | 953.0 | 4.0 | 84.0 | 287.0 | 62.5 |
| $\widehat{\text{PJ}}$ 10 | 5.4 | 192.0 | 111.0 | 7.0 | 1597.0 | 2.0 | 110.0 | 126.0 | 62.7 |
| $\widehat{\text{QJ}}$ 2 | 2.3 | 171.0 | 106.0 | 7.0 | 422.0 | 13.0 | 10.0 | 104.0 | 66.9 |
| J1 | 2.3 | 156.0 | 96.0 | 7.0 | 392.0 | 22.0 | 36.0 | 77.0 | 64.1 |
| J2 | 1.7 | 158.0 | 96.0 | 7.0 | 227.0 | 7.0 | 8.0 | 26.0 | 65.3 |
| JTD | 2.2 | 153.0 | 94.0 | 7.0 | 386.0 | 16.0 | 34.0 | 62.0 | 65.5 |

Table B.57: Results for Monks Set No. 2

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 0.6 | 45.0 | 31.0 | 6.0 | 53.0 | 0 | 0 | 0 | 94.4 |
| ID4 | 1.9 | 42.0 | 28.0 | 6.0 | 1278.0 | 10.0 | 190.0 | 58.0 | 94.4 |
| $\widehat{\text{ID4}}$ | 1.5 | 42.0 | 28.0 | 6.0 | 493.0 | 10.0 | 49.0 | 42.0 | 94.4 |
| ID5 | 3.2 | 42.0 | 29.0 | 5.0 | 1244.0 | 12.0 | 195.0 | 117.0 | 94.9 |
| $\widehat{\text{ID5}}$ | 1.3 | 42.0 | 27.0 | 6.0 | 361.0 | 9.0 | 44.0 | 47.0 | 95.4 |
| ID5R | 3.3 | 42.0 | 28.0 | 6.0 | 1647.0 | 15.0 | 282.0 | 133.0 | 94.4 |
| $\widehat{\text{ID5R}}$ | 1.5 | 45.0 | 29.0 | 6.0 | 586.0 | 10.0 | 90.0 | 57.0 | 93.5 |
| $\widehat{\text{DL}}$ 10 | 1.5 | 50.0 | 33.0 | 6.0 | 379.0 | 8.0 | 48.0 | 168.0 | 92.6 |
| $\widehat{\text{PJ}}$ 10 | 1.0 | 57.0 | 37.0 | 6.0 | 406.0 | 1.0 | 43.0 | 23.0 | 91.2 |
| $\widehat{\text{QJ}}$ 2 | 0.8 | 46.0 | 31.0 | 5.0 | 106.0 | 2.0 | 3.0 | 16.0 | 94.4 |
| J1 | 0.7 | 48.0 | 32.0 | 5.0 | 87.0 | 1.0 | 4.0 | 6.0 | 92.6 |
| J2 | 0.7 | 48.0 | 32.0 | 5.0 | 82.0 | 1.0 | 3.0 | 6.0 | 92.6 |
| JTD | 0.7 | 48.0 | 32.0 | 5.0 | 89.0 | 1.0 | 3.0 | 4.0 | 92.6 |

Table B.58: Results for Monks Set No. 3

### B.2.16 The Effects of Noise

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{DL}$ 0 | 117.1 | 215.8 | 106.9 | 8.0 | 10168.8 | 202.0 | 1714.0 | 8046.4 | 47.6 |
| $\widehat{DL}$ 10 | 41.2 | 213.8 | 106.9 | 8.0 | 6939.5 | 53.0 | 1271.6 | 3047.8 | 48.1 |
| $\widehat{DL}$ 20 | 31.1 | 214.1 | 107.1 | 8.0 | 5370.9 | 53.8 | 1062.8 | 3987.3 | 48.1 |
| $\widehat{DL}$ 30 | 23.9 | 213.9 | 107.0 | 8.0 | 4464.7 | 89.0 | 927.3 | 9178.9 | 48.1 |
| $\widehat{DL}$ 40 | 21.9 | 214.4 | 107.1 | 8.0 | 3707.1 | 30.4 | 812.9 | 3982.2 | 47.8 |
| $\widehat{DL}$ 50 | 22.0 | 215.4 | 107.7 | 8.0 | 3457.3 | 17.9 | 773.4 | 1401.6 | 47.7 |
| $\widehat{DL}$ 100 | 13.4 | 215.7 | 108.0 | 8.0 | 1796.3 | 0 | 539.0 | 0 | 48.0 |

Table B.59: Results for 7-bit LED Set (20% noise)

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{PJ}$ 0 | 196.0 | 215.2 | 106.2 | 8.0 | 9657.6 | 253.2 | 1644.0 | 4990.2 | 47.9 |
| $\widehat{PJ}$ 10 | 39.1 | 212.6 | 106.3 | 8.0 | 11003.7 | 10.2 | 1324.8 | 385.7 | 48.0 |
| $\widehat{PJ}$ 20 | 28.8 | 213.9 | 107.0 | 8.0 | 11114.3 | 4.2 | 1189.8 | 169.1 | 48.0 |
| $\widehat{PJ}$ 30 | 26.8 | 215.8 | 107.7 | 8.0 | 11213.3 | 2.1 | 1064.8 | 84.4 | 47.8 |
| $\widehat{PJ}$ 40 | 22.3 | 215.5 | 107.9 | 8.0 | 11225.7 | 1.1 | 984.2 | 34.1 | 47.9 |
| $\widehat{PJ}$ 50 | 21.4 | 215.1 | 107.7 | 8.0 | 11212.2 | 0.7 | 976.8 | 16.8 | 47.9 |
| $\widehat{PJ}$ 100 | 21.5 | 215.7 | 108.0 | 8.0 | 11248.2 | 0 | 877.6 | 0 | 48.0 |

Table B.60: Results for 7-bit LED Set (20% noise)

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| $\widehat{QJ}$ 1 | 36.7 | 216.4 | 108.6 | 8.0 | 3735.2 | 78.3 | 310.0 | 1237.4 | 48.0 |
| $\widehat{QJ}$ 1.25 | 57.8 | 215.4 | 108.4 | 8.0 | 6371.7 | 112.1 | 576.0 | 2080.3 | 48.0 |
| $\widehat{QJ}$ 1.5 | 84.1 | 216.5 | 108.5 | 8.0 | 8957.0 | 129.8 | 819.0 | 2825.2 | 48.8 |
| $\widehat{QJ}$ 1.75 | 100.1 | 216.6 | 108.6 | 8.0 | 10555.2 | 144.6 | 956.4 | 3220.8 | 47.8 |
| $\widehat{QJ}$ 2 | 106.8 | 216.7 | 108.6 | 8.0 | 11569.3 | 155.0 | 1064.0 | 3389.7 | 47.8 |
| $\widehat{QJ}$ 4 | 119.0 | 215.7 | 108.2 | 8.0 | 13714.5 | 183.0 | 1318.8 | 3806.4 | 47.8 |

Table B.61: Results for 7-bit LED Set (20% noise)

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 10.0 | 216.0 | 108.5 | 8.0 | 306.2 | 0.0 | 0.0 | 0.0 | 47.9 |
| $\widehat{\text{ID5}}$ | 225.7 | 215.2 | 106.2 | 8.0 | 9657.6 | 253.2 | 1644.0 | 4990.2 | 47.9 |
| J1 | 259.1 | 219.9 | 108.3 | 8.0 | 15346.3 | 387.2 | 2653.4 | 6678.3 | 47.8 |
| J2 | 13.7 | 216.8 | 108.5 | 8.0 | 529.0 | 10.8 | 13.1 | 72.0 | 48.1 |
| JTD | 246.5 | 218.4 | 107.9 | 8.0 | 13879.6 | 254.3 | 2338.2 | 5349.9 | 47.9 |

Table B.62: Results for 7-bit LED Set (20% noise)

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 10.2 | 240.0 | 120.5 | 8.0 | 342.6 | 0 | 0 | 0 | 25.2 |
| $\widehat{\text{ID5}}$ | 767.1 | 242.6 | 120.2 | 8.0 | 11979.2 | 407.1 | 1926.8 | 13420.2 | 25.3 |
| J1 | 932.1 | 243.4 | 120.5 | 8.0 | 17161.9 | 747.1 | 2782.1 | 18627.0 | 25.3 |
| J2 | 15.9 | 241.4 | 121.0 | 8.0 | 725.6 | 24.2 | 26.6 | 223.7 | 25.3 |
| JTD | 773.6 | 242.9 | 120.3 | 8.0 | 13956.4 | 421.4 | 2203.8 | 14077.3 | 25.2 |

Table B.63: Results for 7-bit LED Set (30% noise)

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 10.0 | 251.8 | 126.4 | 8.0 | 363.0 | 0 | 0 | 0 | 11.6 |
| $\widehat{\text{ID5}}$ | 1031.7 | 252.9 | 126.3 | 8.0 | 12974.6 | 482.4 | 2009.4 | 16114.8 | 11.4 |
| J1 | 1229.1 | 253.0 | 126.4 | 8.0 | 17663.0 | 944.1 | 2740.0 | 23107.5 | 11.4 |
| J2 | 25.4 | 251.8 | 126.4 | 8.0 | 974.3 | 50.7 | 53.3 | 771.5 | 11.6 |
| JTD | 946.4 | 252.9 | 126.3 | 8.0 | 14005.7 | 492.0 | 2107.5 | 16618.6 | 11.4 |

Table B.64: Results for 7-bit LED Set (40% noise)

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 10.0 | 253.0 | 127.0 | 8.0 | 367.9 | 0 | 0 | 0 | 8.9 |
| $\widehat{\text{ID5}}$ | 1616.3 | 253.6 | 127.1 | 8.0 | 12974.8 | 506.5 | 1955.9 | 19261.0 | 8.8 |
| J1 | 2298.9 | 253.8 | 127.2 | 8.0 | 17776.3 | 1040.0 | 2680.5 | 28245.4 | 8.8 |
| J2 | 13.9 | 253.4 | 127.2 | 8.0 | 596.7 | 16.0 | 18.2 | 122.7 | 8.8 |
| JTD | 1509.9 | 253.6 | 127.1 | 8.0 | 13727.7 | 517.7 | 2011.8 | 20144.7 | 8.8 |

Table B.65: Results for 7-bit LED Set (50% noise)

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 33.0 | 427.8 | 214.4 | 11.9 | 972.1 | 0 | 0 | 0 | 91.6 |
| $\widehat{\text{ID5}}$ | 1796.1 | 535.9 | 266.7 | 12.0 | 10038.4 | 247.9 | 602.2 | 16757.1 | 88.0 |
| J1 | 46.3 | 493.9 | 244.2 | 12.0 | 2577.8 | 90.1 | 97.9 | 332.2 | 90.7 |
| J2 | 41.0 | 481.7 | 237.8 | 12.0 | 1906.9 | 57.7 | 48.4 | 171.5 | 90.6 |
| JTD | 49.1 | 502.9 | 248.6 | 12.0 | 2391.8 | 66.3 | 78.4 | 256.3 | 90.5 |

Table B.66: Results for 11-bit Multiplexor Set (10% noise)

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 35.8 | 684.8 | 342.9 | 12.0 | 1372.4 | 0 | 0 | 0 | 80.5 |
| $\widehat{\text{ID5}}$ | 2476.9 | 764.1 | 376.8 | 12.0 | 13197.5 | 338.3 | 851.5 | 29114.0 | 79.8 |
| J1 | 63.3 | 749.1 | 367.8 | 12.0 | 4521.1 | 180.7 | 257.8 | 964.5 | 80.6 |
| J2 | 46.9 | 749.9 | 367.8 | 12.0 | 2949.9 | 98.2 | 97.7 | 335.2 | 80.4 |
| JTD | 56.2 | 778.4 | 383.0 | 12.0 | 4092.2 | 131.8 | 208.4 | 352.9 | 80.7 |

Table B.67: Results for 11-bit Multiplexor Set (30% noise)

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 40.7 | 835.6 | 418.3 | 12.0 | 1593.0 | 0 | 0 | 0 | 74.4 |
| $\widehat{\text{ID5}}$ | 2244.0 | 861.1 | 422.0 | 12.0 | 16045.9 | 382.9 | 1104.4 | 30215.2 | 74.0 |
| J1 | 88.4 | 884.0 | 433.2 | 12.0 | 6634.0 | 282.6 | 437.3 | 1876.6 | 74.4 |
| J2 | 52.6 | 920.8 | 449.3 | 12.0 | 3363.1 | 106.8 | 105.1 | 362.6 | 74.0 |
| JTD | 65.9 | 870.3 | 426.6 | 12.0 | 5171.7 | 172.3 | 295.9 | 997.8 | 74.4 |

Table B.68: Results for 11-bit Multiplexor Set (50% noise)

| ALG | T | NL | L | MP | EC | R | AR | NR | AC |
|---|---|---|---|---|---|---|---|---|---|
| ID3 | 37.9 | 1262.4 | 631.7 | 12.0 | 1987.3 | 0 | 0 | 0 | 62.4 |
| $\widehat{\text{QJ}}$ 2 | 133.9 | 1200.8 | 597.5 | 12.0 | 18636.8 | 309.7 | 330.5 | 5353.5 | 65.5 |
| J1 | 2257.9 | 1015.5 | 504.5 | 12.0 | 16335.6 | 856.7 | 923.4 | 25020.7 | 76.7 |
| J2 | 100.8 | 1255.4 | 624.5 | 12.0 | 6853.3 | 384.6 | 306.7 | 2615.6 | 64.8 |
| JTD | 410.6 | 855.1 | 423.5 | 12.0 | 8415.0 | 290.9 | 493.5 | 6692.0 | 79.5 |

Table B.69: Results for Parity Set (10% noise)

### B.2.17 The Effects of Pruning

| ALG | NL | L | MP | AC |
|---|---|---|---|---|
| $\widehat{\text{ID5}}$ | 46.8 | 23.9 | 7.2 | 75.2 |
| $\widehat{\text{DL}}$ 10 | 69.4 | 35.1 | 7.9 | 75.3 |
| $\widehat{\text{DL}}$ 20 | 65.8 | 33.3 | 7.9 | 75.4 |
| $\widehat{\text{PJ}}$ 10 | 67.1 | 34.0 | 7.9 | 74.7 |
| $\widehat{\text{PJ}}$ 20 | 67.5 | 34.2 | 8.0 | 75.3 |
| $\widehat{\text{PJ}}$ 30 | 68.5 | 34.7 | 8.0 | 75.5 |
| $\widehat{\text{QJ}}$ 1 | 49.4 | 25.2 | 7.6 | 75.2 |
| $\widehat{\text{QJ}}$ 1.5 | 48.4 | 24.7 | 7.6 | 75.6 |
| $\widehat{\text{QJ}}$ 2 | 48.6 | 24.8 | 7.5 | 75.6 |
| J1 | 55.1 | 28.0 | 7.6 | 75.2 |

Table B.70: Results for 7-bit LED Set with Pruning (10% noise)

| ALG | NL | L | MP | AC |
|---|---|---|---|---|
| $\widehat{\text{ID5}}$ | 60.5 | 30.8 | 8.0 | 53.8 |
| $\widehat{\text{DL}}$ 20 | 82.4 | 41.7 | 8.0 | 53.7 |
| $\widehat{\text{DL}}$ 30 | 81.4 | 41.2 | 8.0 | 54.5 |
| $\widehat{\text{PJ}}$ 10 | 81.8 | 41.4 | 8.0 | 53.7 |
| $\widehat{\text{PJ}}$ 20 | 82.0 | 41.5 | 8.0 | 54.8 |
| $\widehat{\text{PJ}}$ 30 | 81.8 | 41.4 | 8.0 | 53.5 |
| $\widehat{\text{QJ}}$ 1 | 60.4 | 30.7 | 8.0 | 54.3 |
| $\widehat{\text{QJ}}$ 1.5 | 63.8 | 32.4 | 8.0 | 54.2 |
| $\widehat{\text{QJ}}$ 2 | 63.0 | 32.0 | 8.0 | 54.1 |
| J1 | 62.2 | 31.6 | 8.0 | 54.0 |

Table B.71: Results for 7-bit LED Set with Pruning (20% noise)

| ALG | NL | L | MP | AC |
|---|---|---|---|---|
| $\widehat{\text{ID5}}$ | 64.5 | 33.7 | 12.3 | 98.5 |
| $\widehat{\text{DL}}$ 20 | 80.6 | 42.2 | 14.0 | 98.2 |
| $\widehat{\text{DL}}$ 30 | 111.6 | 58.0 | 13.4 | 97.9 |
| $\widehat{\text{PJ}}$ 10 | 125.0 | 64.3 | 13.1 | 97.4 |
| $\widehat{\text{PJ}}$ 20 | 125.0 | 64.3 | 13.1 | 97.4 |
| $\widehat{\text{PJ}}$ 30 | 125.0 | 64.3 | 13.1 | 97.4 |
| QJ 1 | 125.2 | 64.4 | 13.0 | 97.3 |
| QJ 1.5 | 77.2 | 40.1 | 12.1 | 98.6 |
| QJ 2 | 125.0 | 64.3 | 13.1 | 97.4 |
| J1 | 121.4 | 62.5 | 12.9 | 97.6 |

Table B.72: Results for KRKPA7 Chess Set with Pruning

| ALG | NL | L | MP | AC |
|---|---|---|---|---|
| $\widehat{\text{ID5}}$ | 320.6 | 160.8 | 10.8 | 96.0 |
| $\widehat{\text{DL}}$ 20 | 388.6 | 194.8 | 11.1 | 96.3 |
| $\widehat{\text{DL}}$ 30 | 385.0 | 193.0 | 11.2 | 96.6 |
| $\widehat{\text{PJ}}$ 10 | 290.0 | 145.5 | 11.0 | 97.8 |
| $\widehat{\text{PJ}}$ 20 | 258.4 | 129.7 | 11.0 | 97.9 |
| $\widehat{\text{PJ}}$ 30 | 201.8 | 101.4 | 10.6 | 99.0 |
| QJ 1 | 182.4 | 91.7 | 10.6 | 99.0 |
| QJ 1.5 | 162.4 | 81.7 | 9.9 | 99.2 |
| QJ 2 | 145.0 | 73.0 | 9.0 | 99.5 |
| J1 | 262.2 | 131.8 | 10.5 | 98.3 |
| J2 | 262.6 | 131.8 | 10.5 | 98.3 |

Table B.73: Results for 11-bit Multiplexor Set with Pruning

| ALG | NL | L | MP | AC |
|---|---|---|---|---|
| $\widehat{\text{ID5}}$ | 385.2 | 193.1 | 10.5 | 73.5 |
| $\widehat{\text{DL}}$ 20 | 538.0 | 269.5 | 11.0 | 71.2 |
| $\widehat{\text{DL}}$ 30 | 524.4 | 262.7 | 11.0 | 66.7 |
| $\widehat{\text{PJ}}$ 10 | 502.8 | 251.9 | 11.0 | 67.3 |
| $\widehat{\text{PJ}}$ 20 | 362.6 | 181.8 | 10.0 | 61.4 |
| $\widehat{\text{PJ}}$ 30 | 371.0 | 186.0 | 11.0 | 59.0 |
| $\widehat{\text{QJ}}$ 1 | 317.8 | 159.4 | 11.0 | 55.9 |
| $\widehat{\text{QJ}}$ 1.5 | 420.2 | 210.6 | 11.0 | 62.1 |
| $\widehat{\text{QJ}}$ 2 | 301.8 | 151.4 | 11.0 | 67.0 |
| J1 | 388.6 | 194.8 | 10.9 | 92.3 |
| J2 | 388.6 | 194.8 | 10.9 | 92.3 |

Table B.74: Results for Parity Set with Pruning

# Appendix C

The test set used in §3.2.1 has two classes (plus, minus), three attributes with a maximum of four values each and 10% noise:-

- eyes: [blue, brown, green, grey]
- hair: [brown, blond, red, black]
- height: [tall, short]

The membership function (labels examples as positive if true) is as follows:-

$$(eyes = blue \bigvee green) \,\&\, (hair = blond \bigvee red) \,\&\, (height = tall \bigvee short)$$

The actual example set follows:-

```
eyes=grey,hair=blond,height=tall,minus
eyes=blue,hair=black,height=tall,minus
eyes=green,hair=brown,height=short,minus
eyes=brown,hair=red,height=tall,minus
eyes=grey,hair=black,height=tall,minus
eyes=blue,hair=black,height=short,minus
eyes=green,hair=brown,height=tall,plus
eyes=grey,hair=brown,height=short,minus
eyes=brown,hair=black,height=short,minus
eyes=blue,hair=red,height=short,plus
eyes=brown,hair=brown,height=tall,minus
eyes=brown,hair=brown,height=short,minus
eyes=brown,hair=blond,height=tall,minus
eyes=brown,hair=blond,height=short,minus
eyes=blue,hair=red,height=tall,plus
eyes=blue,hair=brown,height=tall,minus
eyes=grey,hair=red,height=tall,minus
eyes=green,hair=red,height=tall,plus
eyes=grey,hair=blond,height=short,minus
eyes=grey,hair=red,height=short,minus
eyes=grey,hair=brown,height=tall,minus
eyes=green,hair=red,height=short,plus
eyes=green,hair=black,height=tall,minus
```

```
eyes=green,hair=blond,height=short,plus
eyes=brown,hair=red,height=short,minus
eyes=green,hair=brown,height=tall,minus
eyes=green,hair=brown,height=short,minus
eyes=green,hair=black,height=short,minus
eyes=brown,hair=black,height=tall,minus
eyes=blue,hair=blond,height=tall,plus
eyes=blue,hair=blond,height=short,plus
eyes=grey,hair=black,height=short,minus
```