# An Incremental, Polynomial-time Algorithm to Induce Disjunctive Normal Form Representations for Propositional Concepts

John k. deVadoss
Department of Computer Science
University of Massachusetts
Amherst, MA 01003.
devadoss@cs.umass.edu

June 30, 1994

**Abstract**

We review the notion of polynomial learnability (Valiant, 1984) for unrestricted DNF and present an alternative notion of the *problem-specific incremental learnability* (Oblow, 1992) of unrestricted DNF. We then present an incremental, polynomial-time algorithm for inducing disjunctive normal form representations, for propositional concepts, from positive and negative examples described in an attribute-based formalism, and show that this satisfies learnability criteria for distribution-specific problems as examples are sampled. Having established a theoretical basis for learnability we then describe a learning system that uses this algorithm to learn DNF from noisy, and inconsistent data, which may be described in terms of both Boolean and symbolic attributes. We empirically compare our system with a number of well known inductive classifier systems on some benchmark problems.

## Contents


**1 Introduction** **3**

**2 Research Goals** **3**

**3 The Problem** **3**
- 3.1 Attribute-based Inductive Learning . . . . . 4
- 3.2 Computational Learning . . . . . 5
  - 3.2.1 Concept Representation (as Boolean formulae) . . . . . 5
  - 3.2.2 Polynomial Learnability . . . . . 6

**4 The Method** **6**
- 4.1 Why is unrestricted DNF not learnable under Valiant's (1984) framework? . 6
- 4.2 Unrestricted DNF can be incrementally learnable in a problem-specific manner 7

**5 The Algorithm** **10**
- 5.1 Illustration . . . . . 13
- 5.2 Symbolic Attributes . . . . . 14
- 5.3 Inconsistent Examples . . . . . 15
- 5.4 Noise and Inexact Concept Definitions . . . . . 16
- 5.5 Minimizing the DNF Representation . . . . . 16
- 5.6 Merging Separately Obtained DNFs . . . . . 17

**6 Complexity Analysis** **18**

**7 Experiments** **19**
- 7.1 Quinlan's Chess Problem . . . . . 20
- 7.2 The 6-bit Multiplexor Problem . . . . . 20
- 7.3 The Tic-Tac-Toe Endgame Problem . . . . . 21
- 7.4 The Monks-2 Problem . . . . . 21
- 7.5 The Promoter Problem . . . . . 22
- 7.6 Empirical Complexity . . . . . 22
  - 7.6.1 Comparative Complexity . . . . . 23

**8 Future Research** **24**
- 8.1 Missing Attribute values . . . . . 24
- 8.2 Space Complexity . . . . . 24
- 8.3 Complementary Concept Versions . . . . . 24

**9 Summary** **24**

# 1 Introduction

Inductive generalization processes form one very popular approach to knowledge acquisition; to derive expert-level knowledge from sets of examples. In particular, concept learning, which Hunt, Marin and Stone (1966) describe as "[the] capacity to develop classification rules from experience," has been a principal area of machine learning research.

We focus on one specific inductive process, namely *attribute-based* induction (Quinlan, 1986). In an attribute-based formalism, we can identify a suitable set of properties or attributes for describing examples. Attributes may be discrete, with a specified set of nominal values, or continuous, with numeric values. Each example belongs to one of a number of mutually exclusive and exhaustive classes, and the learning task may be stated as the search for plausible *propositional* descriptions that explain the given data and are useful for predicting new data (Dietterich & Michalski, 1981).

More specifically, we focus on the *incremental* development of such plausible descriptions that can be described as a disjunct of conjuncts, that is in *Disjunctive Normal Form* (DNF), (Friedman, 1986).

# 2 Research Goals

This research was motivated primarily by the absence of computationally efficient algorithms to induce unrestricted DNF, that is DNF without any restrictions on the size of each conjunct. A secondary goal was to devise a scheme that guarantees learnability when such an algorithm converges. Auxiliary goals include:

- The algorithm should produce an *intelligible* DNF representation for propositional concepts (the *Principle of Comprehensibility*, (Dietterich & Michalski, 1981)).
- The algorithm should produce a *characteristic description* of the concept (Dietterich & Michalski, 1981).
- The algorithm should be able to update the concept by incrementally incorporating information provided by new examples.
- The incremental cost of updating the concept should be much lower than the cost of inducing a new concept from scratch.
- The algorithm should *constructively induce* new compound features or terms.
- The algorithm should handle Boolean and symbolic attributes.
- The algorithm should handle inconsistent training examples.
- The algorithm should execute efficiently in time and space.
- The algorithm should avoid overfitting noise in the examples.
- The algorithm should handle missing attribute values.

# 3 The Problem

The problem that we address has been studied from two related formalisms.

### 3.1 Attribute-based Inductive Learning

We give a rigorous statement of the induction problem. The basis is a universe of *objects* that are described in a zeroeth-order language in terms of a collection of *attributes* (Quinlan, 1986). Each attribute may have either discrete or numeric values, but the attributes used to describe an object must not vary from one object to another. Each object in the universe belongs to one of a number of mutually exclusive and exhaustive classes as opposed to unsupervised learning (Fisher, Pazzani & Langley, 1991) in which appropriate groupings of cases are found by analysis. To simplify the discussion, we assume that there are only two such classes of the concept being learned, on consisting of *positive examples* and the other consisting of *negative examples.*

| Class | Attributes | | | |
|---|---|---|---|---|
| | Outlook | Temperature | Humidity | Windy |
| Negative | sunny | hot | high | false |
| Negative | sunny | hot | high | true |
| Positive | overcast | hot | high | false |
| Positive | rain | mild | high | false |
| Positive | rain | cool | normal | false |
| Negative | rain | cool | normal | true |
| Positive | overcast | cool | normal | true |
| Negative | sunny | mild | high | false |
| Positive | sunny | cool | normal | false |
| Positive | rain | mild | normal | false |
| Positive | sunny | mild | normal | true |
| Positive | overcast | mild | high | true |
| Positive | overcast | hot | normal | false |
| Negative | rain | mild | high | true |

Table 1. A sample training set (Quinlan, 1986).

The learning task has been stated as: given a training set of objects whose classes are known, find a rule for predicting the class of an unseen object as a function of its attribute values (Quinlan, 1990).

Using Boolean formulae as representations for propositional concepts we define the learning task as: given a training set of objects whose classes are known, find a disjunction of conjuncts, that may be used to predict the class of an unseen object as a function of its attribute values.

Table 1 shows a small training (Quinlan, 1986) set of objects whose attribute-values and classes are known.

The above training set may be described by the following minimal DNF:

$$(\textit{outlook} = \text{overcast}) \text{ or } ((\textit{outlook} = \text{rain}) \text{ and } (\textit{windy} = \text{false})) \text{ or } ((\textit{outlook} = \text{sunny}) \text{ and } (\textit{humidity} = \text{normal}))$$

If the examples are consistent, and the attributes are noise-free, it may be possible to obtain (*by some means*) a DNF that correctly classifies each object in the training set.

However, in reality, examples are not always consistent, attributes are noisy, attribute values may be missing, and the resulting DNF may not correctly classify each object in the training set. The motivation is to devise an algorithm that induces a DNF that captures meaningful relationships between an object's class and the values of its attributes under these circumstances such that it moves beyond the training set, to not only correctly classify objects from the training set but other unseen objects as well.

There has been a plethora of research in the area of inducing a classifier for the above induction problem using the Decision Tree formalism (Quinlan, 1983), (Utgoff, 1988), (Utgoff, 1989). However, there has been very little work in the area of directly inducing DNF representations for concepts. In fact, most current work in this area (Pagallo, 1989) has tended to use the decision tree formalism as the basis.

### 3.2 Computational Learning

There has been considerable research in the area of computational learning theory towards developing polynomial-time algorithms for learning concepts from examples in the context of "polynomial learnability" (Valiant, 1984), where the notion of "concept" is identified with that of "Boolean function".

A number of classes of concepts are already known to be polynomially learnable. For example, Valiant (1984) has shown that $k$-CNF (the class of Boolean formulae in conjunctive normal form with at most $k$ literals per clause) is polynomially learnable. It follows that $k$-DNF is also polynomially learnable. It is also known that certain classes of concepts are *not* polynomially learnable, unless $P = NP$. For example, Kearns et al (1987) show that Boolean threshold formulae and $k$-term DNF (Boolean formulae in disjunctive normal form with $k$ terms) for $k \geq 2$ are not polynomially learnable, unless $P = NP$.

We review the notion of polynomial learnability in the context of Boolean formulae in the following section.

#### 3.2.1 Concept Representation (as Boolean formulae)

Boolean formulae can be represented as mappings from objects (assignments) into {0,1} where an assignment is a mapping from a given set $V_n$ of attributes to {0,1}: an assignment gives each variable in $V_n$ a value - either 1 (*true*) or 0 (*false*). If the formula is 1 (*true*) for an object, then that object is a *positive* example of the concept represented by the formula; otherwise it is a *negative* example. More formally, an example of $f$ is a pair (x,$v$) where x is an assignment and $v \in \{0,1\}$. Each variable $x_i$ is associated with two *literals*: $x_i$ itself and its negation $\overline{x}_i$. A *term* is a conjunction of literals and a *clause* is a disjunction of literals. The size of a term or clause is its number of literals. A Boolean formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses. Similarly, a Boolean formula is in *disjunctive normal form* (DNF) if it is a disjunction of terms. A *sample* is a set of examples drawn from some arbitrary distribution defined on the set of input examples. A Boolean formula $f$ is said to be consistent with an example (x,$v$) if $f$(x) = $v$. A Boolean formulae $f$ (concept) is consistent with a sample if it is consistent with each example in the sample.

### 3.2.2 Polynomial Learnability

Informally, a *hypothesis space* $\mathcal{F}$ of functions or concepts is said to be learnable from examples if there exists a learning algorithm such that for any function $f \in \mathcal{F}$, as the algorithm is given more and more examples consistent with $f$, it will converge upon $f$ as the "learned" function.

However, since Boolean functions are defined on a finite domain any class $\mathcal{F}$ of functions is clearly trivially learnable from examples. Thus, to paraphrase Rivest (1987), the interesting questions are:

- is the algorithm *economical*, i.e. does it require *few examples* to distinguish the concept from other alternatives
- is the algorithm *efficient*, i.e. does it require *little computational effort* to identify the the concept from a sufficient number of examples?

Valiant (1984) rigorously formalized the notions by developing the pac-learning formalism with the principle that, the concept that is being learned should closely approximate the concept that is being taught, i.e. that the learned concept should perform well on unseen examples and sample sets. He presupposed the existence of an arbitrary distribution $P_n$ defined on the set of input examples $X_n$. The acquired concept can then be quantified on the quality of its answer by measuring the probability that a new example drawn according to $P_n$ will be incorrectly classified.

More precisely, a set of Boolean functions $\mathcal{F}$ is polynomially learnable if there exists an algorithm $A$ and a polynomial $s(.,.,.)$ such that for all $n$, $\epsilon$, and $\delta$ (where $\epsilon$ is the *accuracy*, and $\delta$ is the *confidence*), all probability distributions $P_n$ on $X_n$, and all functions $f \in F_n$, $A$ will with probability at least 1 - $\delta$, when given a sample of $f$ of size $m = s(\lambda_n, \frac{1}{\epsilon}, \frac{1}{\delta})$ drawn according to $P_n$, output a $g \in F_n$ such that the discrepancy between $f$ and $g$ (i.e. the probability that they differ) $< \epsilon$. Further, $A$'s running time is bounded by a polynomial in $n$ and $m$.

Informally, if $\mathcal{F}$ is polynomially learnable, then it is possible to get "close" to the right answer in a reasonable amount of time, independent of the probability distribution that is used to generate the examples (Rivest, 1987).

Given this framework, it can be shown that unrestricted DNF is *not polynomially learnable*. The problem is *economy*, since any algorithm would have to search an exponential space in order to identify the correct concept.

# 4 The Method

In the following section we look at a well known technique for proving polynomial learnability and rigorously review the non-learnability of unrestricted DNF.

## 4.1 Why is unrestricted DNF not learnable under Valiant's (1984) framework?

In general, proving that a class $\mathcal{F}$ of formulae or functions is polynomially learnable can be very difficult. However, Blumer et al (1987) have devised some elegant methods for doing so. They show that one can prove polynomial learnability by: proving *polynomial-time*

*identifiability* (Rivest, 1987) [1] if the class $\mathcal{F}$ is polynomial-sized [2].

In the case of unrestricted DNF the size $\mid F_n \mid$ of $F_n$ is given by $2^n$ and thus the class of unrestricted DNF is not polynomial-sized. It is therefore known to be not polynomially learnable in the Valiant sense (Valiant, 1984). Informally, independent of the probability distribution that is used to generate the examples, the number of examples that we need to observe in order to learn an unrestricted DNF is given by $2^n$ where $n$ is the number of attributes.

Assuming that there exists a polynomial-time identification algorithm it is now trivial to see why $k$-DNF and $k$-CNF are polynomially learnable. For $k$-DNF and $k$-CNF, the number of examples that we need to observe in order to learn the formulae is bounded by $C_k^n$ which is polynomial in $n$ for fixed $k$. Thus, these problems have generated a great deal of theoretical interest. In practice however, there are very few if any inductive learning algorithms that are direct set-theoretic implementations of Valiant's approach. As pointed out by Oblow (1992), this is in part due to the recent nature of the theory, as well as a result of the apparent inefficiency of the direct set-theoretic implementations as suggested by their worst-case bounds on sample sizes. Blumer et al (1987) have also addressed this concern by pointing out the need to improve sample bounds in both general and specific cases.

In the following section we review Oblow's (1992) approach to bounding the number of examples needed for learning in a problem-specific incremental manner, and use this framework to devise an incremental algorithm for learning DNF that guarantees pac-learnability when it converges, as opposed to the distribution-free pac-learnability.

### 4.2 Unrestricted DNF can be incrementally learnable in a problem-specific manner

While Valiant (1984), Blumer et al (1987) and others have studied the distribution-free bounds on sample complexity in order to guarantee learnability, Oblow (1992) investigates the theory behind a problem-specific incremental (or *psi*) approach to bounding the number of examples for learning. As Oblow notes, using his scheme would allow easier problems to be solved more quickly and with smaller numbers of examples. We briefly review his psi scheme. A more detailed description may be found in Oblow (1992).

In his analysis Valiant (1984) defined a classic urn problem and a Bernoulli trials selection scenario. The urn was assumed to contain $N$ balls of at most $s$ different types, with $s \leq N$. To estimate the contents of the urn, a random sample of balls was drawn from the distribution in order to obtain a representative sample of at least $(1 - \epsilon)$ of the different types of balls. Here, the probability for drawing any ball was taken to be $\frac{1}{N}$. The probability for drawing a particular type $b$ is then given by $\frac{n_b}{N}$ where $n_b$ is the number of balls of type $b$.

This sampling process was described by Valiant as a series of Bernoulli trials, where a

[1] Briefly, a class of formulae $\mathcal{F}$ is said to be polynomial-time identifiable if there exists an *identification algorithm* $A$ and a polynomial $p(\lambda, m)$ such that algorithm $A$ when given the integer $n$ and a sample set $S$ of size $m$, will produce in time $p(\mid F_n \mid, m)$ a function $f \in F_n$ consistent with $S$ (if one exists).

[2] We partition the hypothesis space $\mathcal{F}$ according to the number $n$ of attributes upon which the concept depends. Typically, the learning algorithm will be told at the beginning that the concept to be learned depends on a given set $V_n$ of attributes. Let $F_n$ be the subset of $\mathcal{F}$ that depends on $V_n$, for any integer $n \geq 1$, so that $\mathcal{F} = \cup_{n=1}^{\infty} F_n$ (assuming that each function in $\mathcal{F}$ depends on a finite number of attributes). Then, to prove that the class is polynomial-sized we need to show that $\lg(\mid F_n \mid)$ is polynomial in $n$.

successful trial was defined to be the selection of a ball whose type had not been seen before. In his analysis, each successful trial $j$, with $1 \leq j \leq s$, was assumed to have a probability of success $p_j$ of at least $\epsilon$, the target accuracy for learning (i.e. $p_j \geq \epsilon, \forall j$). A bound on sample complexity was then estimated as the maximum number of examples needed to achieve all $s$ possible successes to within a probability constraint.

Oblow uses Valiant's Bernoulli trials framework to construct a more realistic series of trials where, after each success, the probability for the next success must necessarily *decrease* (since a ball of a certain type with at least one representative is now known to reside in the urn), and the first selection is by definition a success. Using this, he then defines a class of *hard* distributions, the criteria for this being slow rates of decrease in probability after each success, and long waiting times between successes.

With an approximate analysis based on an expected sample size rather than distribution-free considerations, he derives a mean sample complexity that is consistent with Valiant's analysis when the success probability is reduced immediately to $\epsilon$ after the first selection and held constant for all future successes.

Using the above, and the fact that after the $(j-1)$st success, the probability $h(p_j)$ of having a string of $m$ failures in future selections from any probability distribution is given by

$$h(p_j) \equiv (1 - p_j)^m,$$

he proposes a *halting test* such that if no additional successes are found in $m$ future selections, where $m$ is given by

$$m \approx \tfrac{1}{\epsilon} \ln(\tfrac{1}{\delta}),$$

then the selection process may be halted with the assurance that both the $\epsilon$ and $\delta$ conditions for convergence of the learning algorithm would be met.

To implement this halting test, however, one further concern must be addressed. That is, the test is not intended to be used only once in the algorithm. It must be used in sequence after each batch of $m$ examples is sampled. The probability of passing the test after a whole sequence of checks needs to be less than $\delta$ if the desired convergence criteria is to be achieved.

Oblow addresses this concern in an incremental way by changing the batch size at each stage to ensure convergence. Again, using the Bernoulli trials framework, he derives the batch size to be

$$m_j \approx \tfrac{1}{\epsilon} \ln(\tfrac{2^{\beta-1} j^\beta}{(\beta-1)\delta}),$$

where,

$$\beta = 1 + \tfrac{1}{\ln(2j)}.$$

Informally, what he has done is to show that any learning algorithm which uses this halting test will be a batch-wise, incremental learning algorithm that guarantees pac-learnability in a problem-specific manner when it converges i.e. when it has no additional successes in $m_j$ new selections, where, $m_j$ is the batch size at stage $j$.

Let $S$ denote a non-empty input sample of an unknown function $f \in F_n$ and let $j$ denote an auxiliary variable that is initially 0.

BEGIN
$L$ : IF $S$ is empty
THEN GOTO $L'$
ELSE Select $m_j$ examples from $S$
Incrementally construct a set of least specific terms that cover all of the known positive examples without covering any of the known negative examples.
ENDIF
IF there are no misclassifications in the $m_j$ examples
(i.e. there are no additional successes in $m_j$ new selections,
in other words, no new terms have been discovered in the $m_j$ examples)
THEN the concept that has been learned as given by the collection of terms
is guaranteed to satisfy both the error parameter $\epsilon$ and the confidence parameter $\delta$.
ELSE increment $j$ and GOTO L
ENDIF
$L'$ : The concept that has been learned by the algorithm as given by the collection of terms is an approximation to the target concept that is consistent with the observed examples.
END

Figure 1. The Method

Using this framework, we propose an incremental learning algorithm for unrestricted DNF that guarantees pac-learnability when it converges i.e. the learned function $f_L \in F_n$ is equivalent to the target function $f \in F_n$ in the Valiant sense. Otherwise, the function that is learned $f_L$ will be an approximation to the required one $f$.

We assume that there two classes of examples of the concept being learned, i.e. positive and negative, and present a high-level view of the algorithm in Figure 1.

We see that as long as we can construct in polynomial-time, a collection of least specific terms that are consistent with all of the positive examples without being consistent with any of the negative examples, we have an incremental polynomial-time algorithm that will guarantee learnability if it converges, or provide an approximation to the target concept otherwise.

In the following sections we will flesh out the details of our algorithm and show that it is possible to construct incrementally in polynomial-time a set of least specific terms that cover all of the known positive examples without covering any of the known negative examples. Having established a theoretical basis for learnability in a problem-specific manner, we will also show that we can use this algorithm as the basis of a learning system that can handle

noisy, inconsistent data sets that may be described by nominal attributes.

## 5 The Algorithm

The algorithm initially assumes that all unseen examples belong to the class of the first seen example. Until a new example results in a misclassification all the examples are stored. On receipt of the new inconsistent example, the Boolean differences[3] between all of the previously stored examples and the new example is determined. These differences will always be a disjunct of conjuncts (i.e. in DNF form). The significance of these differences lies in the fact that they will *satisfy* (i.e. be consistent with) all of the previously stored examples without being consistent with the new example.

If the new example is a negative example of the concept then if the differences consist of only one conjunct, an arbitrary literal is chosen from that conjunct and a new term is created with this literal (with the negation of this literal, if the new example is a positive example). This allows us to create the least specific term that satisfies all of the previously stored positive examples without satisfying the new positive example (or that satisfies the new positive example without satisfying any of the previously stored negative examples).

However, if the differences consist of more than conjunct, then if the new example is a negative example new terms are created for each one of the conjuncts (if the new example is a positive example then one new term is created by listing the negations of all of the literals present in the differences, without duplicates). Again, this allows us to create the least specific terms that satisfy all of the previously stored positive examples without satisfying the new negative example (or the least specific term that covers the new positive example without satisfying any of the previously stored negative examples).

When a new positive example is misclassified we follow the procedure that we described above to construct another least specific term that satisfies the new example without satisfying any of the previously stored negative examples. However, when a new negative example is misclassified we obtain the Boolean differences between the examples of the offending term and the new negative example. We now restrict (make more specific) the offending term by replacing it with one or more new terms by performing a conjunction of this term with the Boolean differences. That is, we take each conjunct in the differences and create a new term by conjoining this with the literals present in the offending term. We then distribute the examples that were previously stored along with the old term among the new terms i.e. we partition the examples of the old term among the new terms that satisfy them[4].

On further receipt of new positive and negative examples, as long as there is no misclassification, the positive examples are recorded as examples for the appropriate term that satisfies them, and the negative examples are saved separately.

More formally, Let $S$ denote a non-empty input sample of an unknown function $f \in$

[3]The Boolean differences between a collection of examples $X$ and a new example $y$ is given by the disjunction of a number of conjuncts, where each conjunct consists of only those literals that are true for some $x \in X$ and not true for $y$. Duplicate conjuncts, and those that are subsumed by another conjunct, are removed to obtain these differences. For example, the Boolean differences between $\{100, 001, 000\}$ and 011 is given by $x_1 \vee \overline{x}_2$.

[4]Note that each example of the old term is assigned to only one new term even if it is satisfied by more then one new term. This is done by assigning an example to the first new term that satisfies it.

$F_n$, let $j$ denote an auxiliary variable that is initially 0, and let $CLASS$ denote another auxiliary variable that is initially *nil*. In order to facilitate the implementation we assume that the terms that have been learned are ordered in some arbitrary linear fashion without any constraints.

---

1. If $S$ is empty stop.

2. Obtain a new example $x$ from $S$. If this is the first example then set $CLASS$ to be the class label of this example and Goto Step 1.

3. If $j = 0$
   If $CLASS$ = class label of $x$ then save it
   Else Call FIRST-MISCLASSIFICATION. */* we now have the first misclassification */*
   Goto Step 1.
   Else
   For $i = 1$ To $j$
   Check to see whether $TERM(i)$ *satisfies* $x$ i.e. is $TERM(i)$ consistent with $x$?.
   If TERM($i$) satisfies $x$ then Goto Step 4.
   */* no term satisfies the new example */*
   If $x$ is a negative example then record it as one
   Else Call FALSE-NEGATIVE. */* we have misclassified a positive example */*
   Goto Step 1.

4. If $x$ is a positive example then record $x$ as an example of TERM($i$) and Goto Step 1.
   Else Call FALSE-POSITIVE. */* we have misclassified a negative example */*
   Goto Step 3.

The Main Procedure.

---

Procedure FALSE-NEGATIVE
*/* We have misclassified a positive example */*
Set $DIFF$ = the Boolean differences between all the known negative examples and $x$
*/* DIFF will always be a DNF */*
If $\mid DIFF \mid = 1$ then choose an arbitrary literal from the only conjunct in $DIFF$ and create a new term $TERM(J+1)$ with the negation of this literal and record $x$ as its only example.
Else
Collect all the literals present in all of the conjuncts of $DIFF$ and create a new term $TERM(j+1)$ using all of their negations without duplicates and record $x$ as its only example.
Set $j = j + 1$.

Procedure for handling a false negative.

---

Procedure FALSE-POSITIVE
*/* We have misclassified a negative example */*
Set $DIFF$ = the Boolean differences between the examples of TERM($i$) and $x$.
*/* $DIFF$ will always be a DNF */*
Remove the $TERM(i)$ from the list of terms.
If $\mid DIFF \mid = 1$ then
  Choose an arbitrary literal from the only conjunct in $DIFF$ and conjoin it
  with $TERM(i)$ and insert this term in position $i$.
  Record all the examples of $TERM(i)$ as the examples of this new term.
Else
Set $k = j$ and Set $j = j - 1$.
    ForEach Conjunct $q$ in $DIFF$ Do
        Create a new term by using the literals in $q$ and those in $TERM(i)$ and
        insert it as $TERM(k)$.
        Record as examples of $TERM(k)$ all of the examples of $TERM(i)$
        that are satisfied by $TERM(i)$.
        Set $k = k + 1$.
    Set $j = j+ \mid DIFF \mid$.

Procedure for handling a false negative.

---

Procedure FIRST-MISCLASSIFICATION
*/* We have misclassified a new example for the first time */*
Set $DIFF$ = the Boolean differences between all the known examples and $x$
*/* $DIFF$ will always be a DNF */*
If $CLASS$ = "$-$"
*/* i.e. all of the previously stored examples are negative */*
  If $\mid DIFF \mid = 1$ then choose an arbitrary literal from the only conjunct in $DIFF$ and
  create a new term $TERM(1)$ with the negation of this literal and record $x$ as its only
  example.
  Else
      Collect all the literals present in all of the conjuncts of $DIFF$ and create a new term
      $TERM(1)$ using all of their negations without duplicates and record $x$ as its only
      example.
  Set $j = 1$.
Else
*/* i.e. all of the previously stored examples are positive */*
    If $\mid DIFF \mid = 1$ then
      Choose an arbitrary literal from the only conjunct in $DIFF$ and
      insert this as $TERM(1)$.
      Record all of the previously stored positive examples as examples of this new term,

and store the new negative example as the only known negative example.
Set $j = 1$.
Else
Set $k = 1$.
ForEach Conjunct $q$ in $DIFF$ Do
Create a new term by using the literals in $q$ insert it as $TERM(k)$.
Record as examples of $TERM(k)$ all of the previously stored examples that are satisfied by it.
Set $k = k + 1$.
Set $j = j + k$.

Procedure for handling the first misclassification.

---

### 5.1 Illustration

In order to illustrate the functioning of the algorithm let us observe how it learns the Boolean concept:

$$\overline{x}_1 x_2 \vee x_1 \overline{x}_3 \vee x_1 x_2 x_4.$$

We assume that there are 5 attributes, and that for this example all the $2^5$ examples are given in order from 00000 to 11111.

On receipt of the first example (- 00000) the algorithm assumes that all unseen examples are *negative* i.e. the Most Specific Version of the concept.

With this version of the concept all succeeding examples until (+ 01000) are classified correctly and are stored as negative examples. However (+ 01000) results in a misclassification since it is a positive example. We now determine the Boolean differences between 01000 and all of the previously stored negative examples (- 00000), (- 00001), (- 00010), (- 00011), (- 00100), (- 00101), (- 00110), and (- 00110). The difference is found to be $\overline{x}_2$, and the first term in the DNF is created with $x_2$ as its only literal, and with 01000 as its only example i.e. we have just created the least specific term that satisfies the new positive example without satisfying any of the previously stored negative examples. Intuitively, we have just generalized the concept to cover a new positive example. At this time the concept is characterized by:

$$x_2.$$

With this version of the DNF all succeeding examples until (+ 10000) are classified correctly and are stored as examples of $x_2$. However, since the only existing term does not satisfy 10000, it results in a misclassification. We now determine the Boolean differences between 10000 and all of the previously stored negative examples. This is found to be $\overline{x}_1$, and we now create another least specific term with $x_1$ as its only literal and store 10000 as its only example. Again, we have generalized the concept to cover a new positive example. At this time the concept is characterized by:

$$x_1 \vee x_2.$$

With this version of the DNF all succeeding examples until (- 10100) are classified correctly and are stored as examples of $x_1$. However, 10100 results in a misclassification on the term $x_1$. Intuitively, we now need to restrict the concept to prevent it from covering the new negative example. We determine the Boolean differences between the examples of $x_1$ (the offending term) and 10100. This is found to be $\overline{x}_3$. We now restrict the term $x_1$ by conjoining it with the literal $\overline{x}_3$. The misclassified example (- 10100) can now be stored as a negative example. At this time, the concept is characterized by:

$$x_1\overline{x}_3 \vee x_2.$$

The succeeding examples until (- 11100) are classified correctly. However, 11100 is classified as a positive example by the term $x_2$. Again, we need to restrict this term in order to prevent it from covering a new negative example. We now determine the Boolean differences between the stored examples of $x_2$ and 11100. This is found to be $\overline{x}_1$. We now restrict the term $x_2$ by conjoining it with the new literal $\overline{x}_1$. The misclassified example (- 11100) can now be stored as a negative example. At this time, the concept is characterized by

$$x_1\overline{x}_3 \vee \overline{x}_1x_2.$$

The next example (- 11101) is classified correctly. However, (+ 11110) results in a misclassification since it is not satisfied by any term in the DNF. Again, we need to generalize the concept to cover this new positive example. We determine the Boolean differences between all of the stored negative examples and 11110. This is found to be $\overline{x}_2 \vee \overline{x}_4$. We now create another least specific term with $x_2$ and $x_4$ as its literals and record 11110 as its only example. At this time, the concept is characterized by:

$$x_1\overline{x}_3 \vee \overline{x}_1x_2 \vee x_2x_4.$$

This correctly classifies the last example (+ 11111).

We note that the algorithm has learned a more compact representation of the concept than the one we started out with, and that it has ignored the irrelevant attribute $x_5$.

### 5.2 Symbolic Attributes

In order to construct the terms of the DNF as described above, each example must be represented as a sequence of 0's and 1's along with its classification. For many-valued symbolic attributes, we dynamically encode each attribute-value pair as a propositional attribute which is *true* (1) if and only if the attribute has taken on the particular value in the example, and *false* (0) otherwise (Hampson & Volper, 1986). This scheme makes it possible to describe examples by a mix of both boolean and symbolic attributes.

In fact, in the context of decision trees, Mooney et al (1989) have observed that mapping many-valued attributes to two-valued propositional attributes consistently increases classification accuracy.

### 5.3 Inconsistent Examples

Two examples are inconsistent if they are described by the same attribute-values but have different class labels. When inconsistent examples occur we always assume the class label of the newer example to be correct. This allows the algorithm to track trends in the concept. We illustrate this with an example.

Let us consider the Boolean concept $\overline{x}_1 x_2 \vee x_1 \overline{x}_2$. We assume that the training examples are from 00 to 11 and will be given in that order. As before, each training example has a concept value (either +, or -).

On receipt of the first example (- 00), the algorithm assumes that no example is positive, and records is as a negative example. The next example that is received is (+ 01) and this results in a misclassification. The Boolean differences between the two examples are determined and a new term $x_2$ is created. At this time, the only term that is known is $x_2$. The next example that is received is (+ 10), and again this results in a misclassification (the term $x_2$ does not cover this new example). The Boolean difference between the negative example (- 00) and the new positive example (+ 10) is determined to be $x_1$ and a new term is created. At this time the only terms that have been learned are $x_2$ and $x_1$. The next example that is received is (- 11). This results in a misclassification on the term $x_2$. The Boolean difference between (+ 01) (the only recorded example for $x_2$) and (- 11) is determined to be $\overline{x}_1$ and the old term $x_2$ is constrained by the addition of the literal $\overline{x}_1$ resulting in two terms $\overline{x}_1 x_2$ and $x_1$. The next example (- 11) results in a misclassification on the term $x_1$. The Boolean difference between (+ 10) (the only recorded example for $x_1$) and (- 11) is determined to be $\overline{x}_2$ and the old term $x_1$ is constrained by the addition of the literal $\overline{x}_2$ resulting in two terms $\overline{x}_1 x_2$ and $\overline{x}_1 x_2$. The example (- 11) is now stored as a negative example since it is not satisfied by any term.

The concept at this time as described by the terms is:

$$x_1 \overline{x}_2 \vee \overline{x}_1 x_2.$$

Now let us suppose that the algorithm is given the new inconsistent example (- 01). This negative example will be satisfied by the term $\overline{x}_1 x_2$. The Boolean difference between the old example (+ 01) and the new negative example (- 01) is found to be null. Therefore the old term $\overline{x}_1 x_2$ is removed from the list of terms and the example (- 01) is now by default stored as a negative example of the concept. This is because the only available term $x_1 \overline{x}_2$ does not satisfy this example.

The concept at this time as described by the terms is:

$$x_1 \overline{x}_2,$$

with (+ 10) as its only recorded example.

Again, let us suppose that the algorithm is given the new example (+ 01). This positive example will not be satisfied by any term. The Boolean difference between this example and the available negative examples (- 00), (- 11), and (- 01) is determined to be $\overline{x}_1 x_2$ and a new term is created. The old inconsistent example (- 01) is discarded and the new positive example (+ 01) is recorded as the only example of the new term.

The concept at this time as described by the terms is:

$$x_1\overline{x}_2 \vee \overline{x}_1 x_2.$$

### 5.4 Noise and Inexact Concept Definitions

In the real-world it is often impossible to precisely characterize or discriminate concepts. Under such circumstances, it has been determined that simple, approximate concepts are often more valuable than overspecified concept definitions that "fit the noise." In the case of classification knowledge expressed as decision trees or rules, Breiman et al (1984), and Quinlan (1987) have obtained improved performance on unseen cases by "pruning".

In this context, following Rissanen's Minimum Description Length Principle (1983), we contend that the number of bits required to encode a term should never exceed the number of bits needed to indicate explicitly the positive examples that are examples of that term, and are therefore covered by that term. The reasoning resembles that used by (Quinlan, 1990) and (Quinlan & Rivest, 1989).

This criterion may be used to decide whether:

- a term should be extended further even though in its current state it satisfies a new negative example, and whether
- a new term should be created, even though the current collection of terms do not satisfy a new positive example.

In the context of incremental learning, however, it is not possible to use the above rules since it is always possible that a term that is currently known to violate the Minimum Description Principle may later be found to satisfy the Principle by virtue of covering newer examples and assimilating them in its example set.

Hence, we use a technique that we call *shadow pruning* wherein, for training, we use all of the available terms irrespective of their quality as quantified by the above means, while for classification we use only those terms that satisfy the MDL Principle (i.e. only the terms that satisfy the MDL Principle are considered to characterize the concept).

This is a simplistic approach to the problem of inexact concept definitions and we are currently investigating alternative approaches. In particular, in the context of CNF representations, Holte et al (1989) note that *small disjuncts*, i.e. disjuncts that cover a few training examples, are usually less accurate than the larger ones, and offer an approach to reduce the risk of using such disjuncts by changing bias. On the other hand, Quinlan (1991), using a Bayes-Laplace model, shows that size alone is not an adequate basis for predicting the error rate of a disjunct and hence advocates the need for different strategies. We are currently evaluating the applicability of the above results to DNFs.

### 5.5 Minimizing the DNF Representation

Most incremental algorithms are inherently subject to the effect of the order of the training examples on the nature of the concept learned. The exceptions would include algorithms that maintain multiple versions of the concept (in some description language) (Mitchell, 1978) and those that incrementally restructure the entire concept using all of the information that is known at each stage (Utgoff, 1989).

The size of the DNF that is learned by our algorithm is certainly dependent on the "nature" of the input examples. If the algorithm initially receives examples that characterize

or discriminate the concept definition more precisely, then we can learn more succinct representations. However, the classification performance is not adversely affected, although we might require more examples in order to satisfy pac-learnability as defined in Section 3.2.2.

The problem of finding the smallest Boolean formula consistent with a set of training examples has long been known to be NP-complete (Gold, 1978). However, we can use heuristics to attempt to yield smaller DNFs, in terms of both the number of gates (terms), as well as the number of gate inputs (literals in each term) (Friedman, 1986).

In the algorithm of Section 5 whenever a new term is created, either from scratch, or by restricting an already available term, we can determine:

- whether the new term subsumes (or is subsumed by an existing term), and
- whether the new term can be combined with an existing term.

In the first case, the term that is subsumed can be removed and its examples can be recorded with those of the subsumee.

In the second case, we are trying to determine whether two terms can be replaced by their intersection i.e. by a more general term. The caveat here is that we have to determine whether the new term is *more general* than the prior two terms i.e. we have to determine whether this new term covers some negative examples that the two previous terms did not. This can be easily accomplished by examining the available negative examples and checking to see whether the new term satisfies one or more of those.

It will be shown in Section 6 that the above operations can be performed in time linear in the number of attributes $v$, as well as the number of terms $n$.

We tried to evaluate the effectiveness of these heuristics by comparing the sizes of the DNFs that were obtained by our algorithm with that of *espresso* (Brayton, Hachtel & Sangiovanni-Vincentelli, 1984), a well known Boolean minimization algorithm for VLSI design, on some VLSI benchmarks. Although the sizes of our two-level circuits were significantly larger than those of *espresso*, we were able to obtain our results under execution times that were significantly less than those of espresso. Thus, our heuristics seem to be a viable alternative for adaptive, incremental synthesis of two-level Boolean circuits that may be used to design Application Specific Integrated Circuits (ASICs) or Programmable Logic Arrays (PLAs).

### 5.6 Merging Separately Obtained DNFs

Most real-world induction tasks require learning from a large variety and number of training examples. Merging the concepts learned from different data sets may be a desirable capability in this regard. One may even consider analyzing a number of data sets in parallel if necessary.

Whereas decision trees and decision lists do not allow for computationally efficient methods for merging two or more concepts, the use of DNF representations for concepts allows for a very simple and polynomial-time technique to merge two or more concepts.

The union of two or more DNF representations takes time linear in the total number of terms. Then, we may use the heuristics described in Section 5.5 to reduce the number of

terms and/or the number of literals. The computation is bounded by $O(n^2v)$ where $n$ is the number of terms present in the resulting union, and $v$ is the number of attributes.

## 6 Complexity Analysis

The following analysis proceeds with the assumptions that the number of examples, both positive and negative, that have been incorporated is given by $N$, that there are currently $n$ terms (conjuncts), and that the number of attributes is given by $\mid \mathit{Attributes} \mid = v$. As in Section 5 we assume that the terms in the DNF are ordered in some linear fashion.

In the worst case $n$ is equal to the number of *positive* examples (which corresponds to the notion of a trivial polynomial-time identification algorithm). However, in the average case $n$ can be assumed to be equal to $N^{1/2}$, where each term can be assumed to cover $N^{1/2}$ examples. Hence, $n$ is bounded by $O(N)$.

1. In the case where a new example is a *positive example* and is correctly satisfied by an existing term, or is a *negative example* and is not satisfied by any existing term and is therefore trivially satisfied, the computational time is given by $O(nv)$ since there are at most $n$ terms to be verified for satisfiability, and each term has at most $v$ literals that have to verified for satisfiability. Hence, the computation is bounded by $O(Nv)$.

2. In the case where a new *positive example* is not satisfied by an existing term, the computation is bounded by:

   - the amount of time taken to determine that the new example is not satisfied by any existing term. This is given by $O(nv)$ since there are $n$ terms that have to checked, and each term has at most $v$ literals that have to be verified for satisfiability.
   - the amount of time taken to determine the differences between the negative examples that have been seen so far and this new positive example. This is given by $O(Nv)$ since there could be at most $N$ negative examples that have been seen so far, assuming that this is the first positive example of the concept, and since each negative example will contain at most $v$ literals. In the average case however, we can expect the number of negative examples that have been seen so far to be $N/2$. In any case, the computation is bounded by $O(Nv)$.
   - the amount of time taken to construct a new term using the differences that have been determined and to make this new positive example an example of the new term. This is bounded by $O(v)$ since there could be at most $v$ literals in the new term.
   - the time required to determine if any existing term subsumes this new term (and equivalently to determine whether this term subsumes an existing term) or whether this new term can be combined with an existing term.

     If the new term is indeed subsumed by an existing term, or subsumes an existing term, or can be combined with an existing term, then the examples of the two older terms have to be assigned to the resulting term (either the subsumee, or the intersection of the two older terms). Further, the terms that have been found to be more specific than necessary have to be removed.

To determine subsumption or intersection the computation is given by $O(nv)$ since there are currently $n$ terms and each term has at most $v$ literals. The computation required to incorporate the examples of the existing terms into those of the new term is given by $O(N)$ since there could potentially be $N$ examples that need to be incorporated.

In any case, the total computational time is bounded by $O(Nv)$.

3. Conversely, where a new *negative example* is satisfied by an existing term there are three possible scenarios:

   In the best case, this new negative example is satisfied by only one term. The computational time is give by:

   - the amount of time taken to determine that this is in fact satisfied by an existing term. This is bounded by $O(nv)$.
   - the time taken to find the differences between the examples in the offending term and this new example. This is bounded by $O(Nv)$.
   - the time taken to construct a new term(s) using these differences and to partition the examples of the old term among the new term(s). This is given by $O(v) + O(nl)$, where $l$ is the number of new terms that have been created.
   - the time taken to determine whether this new term(s) is subsumed by an existing term or subsumes an existing term, or can be combined with an existing term, and if necessary the computation required to assign the examples of the old term(s) to the new term. This is bounded by $O((n + l)v)$.

     Hence, the computation is bounded by $O(Nv)$.

   In the worst case, this new negative example is satisfied by every term present. Analyzing similarly to the above, the computational time in this case is given by the function by $\sum_{m=1}^{m=n} C(m)$, where $C(m) = O((n-m+1)v)+O(Nv)+O(Nl)+C(m-1)+O((n+l)v)$. This is bounded by $O(N^2v)$. In the average case, gain using the $C(m)$ we have just defined we would expect the computational time to be bounded by $O(Nvlog(N))$.

   We note that these complexity measures correspond to *comparison operations*, and that the practical efficiency of these operations is closely tied to their implementation efficiency.

   The space complexity is bounded by $\theta(N)$ where $N$ is the total number of positive and negative examples that have been examined.

# 7 Experiments

We compare the predictive performance of our algorithm with that of C4.5 (Quinlan, 1993), ITI (Utgoff, 1994), and CDL2 (Shen, 1992) on a reasonably representative set of benchmark domains (Zheng, 1993).

While C4.5 is a one-shot learning system and ITI is an incremental learning algorithm, both induce classifiers in the form of decision trees. On the other hand, CDL2 is an incremental learning algorithm that induces decision lists. C4.5 is a derivative of the well known

ID3 (Quinlan, 1986), and ITI is a derivative of ID5R (Utgoff, 1989), an incremental version of ID3.

Rather than trying to estimate the reliability of a classification model by dividing the data into a training and test, building the model using only the training set and examining its performance on the unseen test cases we follow Quinlan's (1993) suggestion by performing a 10-way cross-validation test on a number of known domains. This method allows us to obtain a more robust estimate of the accuracy on unseen cases.

The method that we follow is to divide the available data into 10 blocks so as to make each block's number of cases and class distribution as uniform as possible. We then build 10 different classification models, in each of which one block is omitted from the training data, and the resulting model is tested on the cases in the omitted block. Thus, each case appears in exactly one test set. Quinlan suggests that the average error rate over the unseen test sets is a good, albeit pessimistic, predictor of the error rate of a model.

### 7.1 Quinlan's Chess Problem

Quinlan's (1983) Chess End-game Problem has long been a whetstone for aspiring inductive learning algorithms and we stick to tradition by conducting the first of our experiments with this problem.

The examples in this task are configurations of chess end games. There are 39 binary attributes, and the task is to learn the concept of *win* from a collection of 551 examples. We present below comparative results with C4.5, CDL2, and ITI on a 10-way cross-validation.

| Algorithm | 10-way Cross-validation Results | |
|---|---|---|
| | Average number of decisions | Average unseen Error Rate |
| C4.5 | 80.8 | 8.00% |
| CDL2 | 45.7 | 9.62% |
| ITI | 70.0 | 8.04% |
| DNF | 36.7 | 6.18% |

Table 2. Quinlan's Chess Problem (Quinlan, 1983).

In this comparison, and all the others that follow, we note that the decisions for ITI and C4.5 are single tests, whereas for CDL2 and our algorithm they are multiple tests. A uniform comparison on the basis of just the number of decisions is therefore not fair.

### 7.2 The 6-bit Multiplexor Problem

In this task (Barto, 1985), each example is described by a series of bits, the first $x$ of which constitute an address (i.e. from 0 to $2^x$ - 1), followed by one data bit for each address. There are two classes, *positive* and *negative*, and an example is a positive example if the data bit corresponding to the address is 1.

This has been a pathological task for decision tree classifiers since all selection criteria usually seem to lead to a initial division that tests a data bit rather than an address bit (Quinlan, 1993).

| Algorithm | 10-way Cross-validation Results | |
|---|---|---|
| | Average number of decisions | Average unseen Error Rate |
| C4.5 | 30.6 | 30.00% |
| CDL2 | 14.3 | 17.12% |
| ITI | 19.7 | 14.29% |
| DNF | 8.8 | 9.29% |

Table 3. The 6-bit Multiplexor Problem (Barto, 1985).

The results of a 10-way cross-validation on the 6-bit multiplexor task with 64 examples are show in the table above.

### 7.3 The Tic-Tac-Toe Endgame Problem

The Tic-Tac-Toe Endgame problem presented by Aha(1991) encodes the complete set of possible board configurations at the end of Tic-tac-toe games, where $X$ is assumed to have played first. The target concept is a *win for X*. The UCI database notes that basic decision tree algorithms have had problems with this task, whereas rule-based and example-based learning algorithms have experienced success. There are 9 attributes, with three possible values each, $X$, $O$, or *blank*. There are no missing attribute values. We present below comparative results with ITI and C4.5 on a 10-way cross-validation. We do not present results for CDL2 since it can handle only binary attributes.

| Algorithm | 10-way Cross-validation Results | |
|---|---|---|
| | Average number of decisions | Average unseen Error Rate |
| C4.5 | 185.8 | 15.2% |
| ITI | 75.8 | 7.81% |
| DNF | 34.7 | 3.34% |

Table 4. The Tic-tac-toe Endgame Problem (Aha, 1991).

### 7.4 The Monks-2 Problem

The Monk's Problems are artificial tasks designed to test learning algorithms (Thrun, 1991). Collectively, they are a collection of three binary classification problems over a six-attribute discrete domain. The concept underlying the Monks-2 Problem is given by:

$$(\text{attribute } n = 1) \text{ for } \textit{exactly two} \text{ choices of } n.$$

Decision trees and rules usually do not contain tests of the above nature. However, it is potentially possible that a learning algorithm that constructively induces compound terms may find this to be a viable problem. We present below the results of a 10-way cross-validation over the set of 432 examples. We again do not present the results for CDL2 since it cannot handle symbolic attributes.

| Algorithm | 10-way Cross-validation Results | |
|---|---|---|
| | Average number of decisions | Average unseen Error Rate |
| C4.5 | 148.3 | 49.3% |
| ITI | 46.5 | 1.82% |
| DNF | 31.1 | 15.50% |

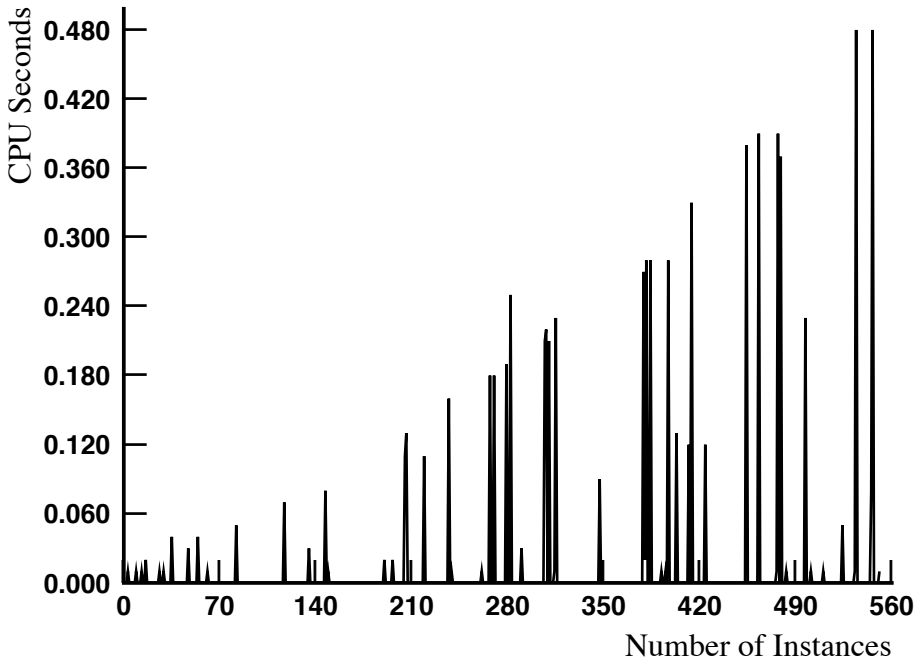


Figure 2. Quinlan's Chess Problem (1983)

Table 5. The Monks-2 Problem (Thrun, 1991).

### 7.5 The Promoter Problem

The Promoter Problem is a relatively unknown domain for inductive learning systems. However, it is another one of the domains that has been found to be a reasonably good benchmark (Zheng, 1993) for such systems. There are 57 attributes, each assuming 1 of 4 possible values, and 106 examples, which are divided into two classes.

We present below the results of a 10-way cross-validation on this domain. We again do not present the results for CDL2 since it cannot handle symbolic attributes.

| Algorithm | 10-way Cross-validation Results | |
|---|---|---|
| | Average number of decisions | Average unseen Error Rate |
| C4.5 | 31.4 | 20.60% |
| ITI | 11.3 | 26.36% |
| DNF | 5.5 | 23.55% |

Table 6. The Promoter Problem.

### 7.6 Empirical Complexity

The worst-case analysis in Section 6 is quite liberal and it is very likely that the average empirical performance is much better. In order to obtain information about the average-case complexity, we examine the algorithm's incremental learning costs on two of the above problems, the Chess End Game Problem, and the Tic-tac-Toe Problem. These results are shown in Figures 2 and 3.

These figures seem to reinforce the opinion that the results in Section 6 could be tightened.

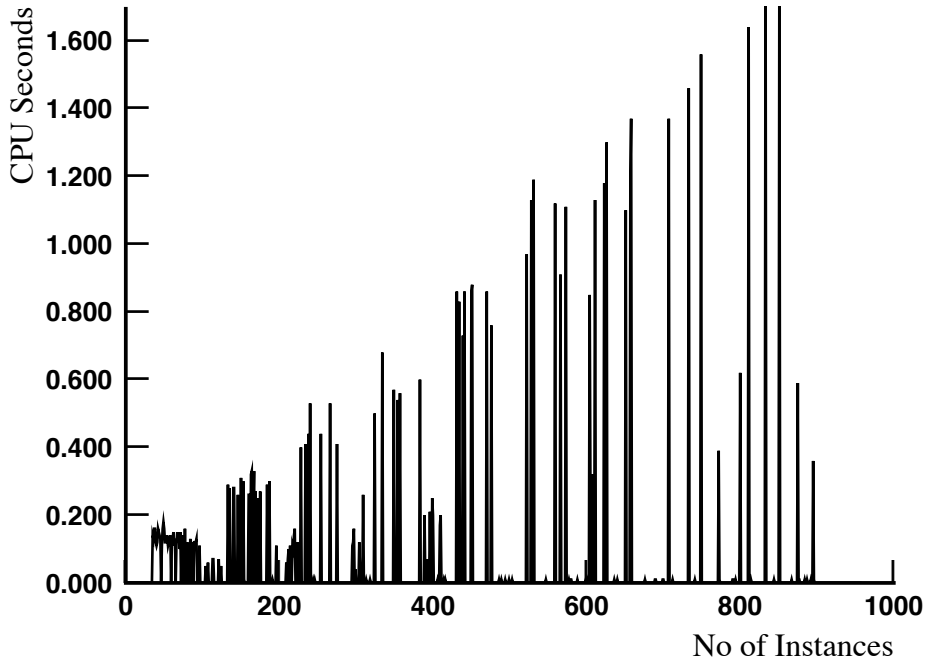


Figure 3. Aha's Tic-Tac-Toe Problem (1991)

### 7.6.1 Comparative Complexity

We have compared the predictive performances of our algorithm with that of C4.5 (Quinlan, 1993), ITI (Utgoff, 1994) and CDL2 (Shen, 1992) in the previous sections. Here we try to estimate the comparative computational complexities of these algorithms. We assume that $n$ is the number of examples, and that $v$ is the number of attributes. Since complexity analyses for C4.5 and ITI are not available, we use Utgoff's results (1989) for ID3 and ID5R, their respective predecessors.

Utgoff estimates that ID3 takes $O(nv^2)$ additions and $O(2^v)$ multiplications where $n$ is the number of examples, and $v$ is the number of attributes. In the same work he states that ID5R takes $O(nv2^v)$ additions and $O(2^v)$ multiplications. Shen (1992) states that CDL2 takes $O(n^3v)$ comparison operations. These measures are shown in Table 7.

| Algorithm | Complexity | | |
|---|---|---|---|
| | Additions | Multiplications | Comparisons |
| ID3 | $O(nv^2)$ | $O(2^v)$ | |
| ID5R | $O(nv2^v)$ | $O(2^v)$ | |
| CDL2 | | | $O(n^3v)$ |
| DNF | | | $O(n^2v)$ |

Table 7. Comparative Complexity Measures.

Theoretically, we perform better than CDL2 by a polynomial degree. We also differ from ID3 and ID5R in that we do not have any exponential components.

# 8 Future Research

The algorithm described in Section 5 satisfies all of our goals except the last one. We describe ongoing work towards achieving this goal as well as other enhancements to the algorithm in this section.

## 8.1 Missing Attribute values

In practice data often has missing attribute values and the induction algorithm has to learn from training examples that are incomplete, as well as classify test examples that are incomplete.

In the context of decision trees, Kononenko et al (1984), Quinlan (1986) and Cestnik et al (1987) have explored the issues involved with trying to fill in an unknown value by utilizing information provided by context. We are currently investigating the implementation of a similar functionality in our algorithm.

## 8.2 Space Complexity

A learning algorithm is *behaviorally incremental* (Shen, 1992) if it is incremental in behavior (i.e. it can process examples one at a time), but it requires to remember some or all previous examples.

Our algorithm (like ID5R, and CDL2) is certainly behaviorally incremental in that we need to remember all previous examples.

One of our goals is to investigate statistical techniques that may allow us to reduce the space complexity by alleviating the need to store all the information that is provided by a new example.

## 8.3 Complementary Concept Versions

In Section 5.5 we presented heuristics that minimize the effect of the order of the training examples on the size of the DNF (in terms of both the number of terms, as well as the number of literals in each term).

Another mechanism that we may use to reduce the number of terms and/or the number of literals is to learn both the concept as well as its negation. From Section 5 we see that we can maintain the DNF representation of the concept as well as its negation in polynomial time. We can then choose the smaller, and therefore potentially better (Rissanen, 1983) concept version.

Another advantage of this approach lies in the fact that using the results of Section 4.2 we can now choose the concept version that converges first.

# 9 Summary

We have reviewed the learnability of unrestricted DNF under Valiant's (1984) framework, and presented an alternative notion of the problem-specific incremental learnability of unrestricted DNF using Oblow's (1992) framework. We have then described an incremental polynomial-time algorithm for learning unrestricted DNF that satisfies learnability criteria on a distribution-specific basis when it converges.

Using this algorithm as the core, we have described a new learning system that induces

DNF representations for propositional concepts described in an attribute-value formalism, from noisy, and inconsistent data sets consisting of both positive and negative examples.

Using 10-way cross-validation results on five benchmark problems we have demonstrated that the predictive performance of our algorithm is better than that of CDL2, ITI and C4.5 on three of the five problems, and that it is better than that of C4.5 in one of the remaining two problems, and better than that of ITI on the last one.

## Acknowledgement

I am indebted to Prof. Paul E. Utgoff for all of his help. Without his encouragement, his enthusiasm and most of all, his patience, this work could not have been accomplished.

## References

Aha, D. W. (1991). Incremental constructive induction: An instance-based approach. *Machine Learning: Proceedings of the Eighth International Workshop* (pp. 117-121). Evanston, IL: Morgan Kaufmann.

Barto, A. G. (1985). Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology, 4*, 229-256.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1987). *Learnability and the Vapnik-Chervonenkis dimension*, (UCSC-CRL-87-20), Santa Cruz, CA: University of California.

Brayton, R. K., Hachtel, G. D., & Sangiovanni-Vincentelli, A. L. (1984). *Logic minimization algorithms for vlsi synthesis.* Hingham, MA: Kluwer .

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees.* Belmont, CA: Wadsworth International Group.

Cestnik, B., Kononenko, I., & Bratko, I. (1987). ASSISTANT 86: A knowledge-elicitation tool for sophisticated users. In Bratko & Lavrac (Eds.), *Progress in Machine Learning.* Wilmslow, UK: Sigma Press.

Dietterich, T., & Michalski, R. (1981). Inductive learning of structural descriptions. *Artificial Intelligence, 16*, 257-294.

Fisher, D. H., Pazzani, M. J., & Langley, P. (1991). *Concept formation: Knowledge and experience in unsupervised learning.* San Mateo, CA: Morgan-Kaufmann.

Friedman, A. D. (1986). *Fundamentals of logic design and switching theory.* Rockville, MD: Computer Science Press.

Gold, E.M. (1978). Complexity of automaton identification from given data. *Info. Control, 37*, 302-320.

Hampson, S. E., & Volper, D. J. (1986). Linear function neurons: Structure and training. *Biological Cybernetics, 53*, 203-217.

Holte, R. C., Acker, L. E., & Porter, B. W. (1989). Concept learning and the accuracy of small disjuncts. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 813-818). Detroit, Michigan: Morgan Kaufmann.

Hunt, E., Marin, J,, & Stone, P. (1966). *Experiments in induction.* Academic Press Inc..

Kearns, M., Li, M., Pitt, L., & Valiant, L. (1987). On the learnability of boolean formulae. *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing* (pp. 285-295). New York, NY.

Kononenko, I., Bratko, I., & Roskar, E. (1984). *Experiments in automatic learning of medical diagnosis rules*, (unpublished), Ljubljana, Yugoslavia: Jozef Stefan Institute.

Mitchell, T. M. (1978). *Version spaces: An approach to concept learning.* Doctoral dissertation, Department of Electrical Engineering, Stanford University, Palo Alto, CA.

Mooney, R., Shavlik, J., Towell, G., & Gove, A. (1989). An experimental comparison of symbolic and connectionist learning algorithms. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 775-780). Detroit, Michigan: Morgan Kaufmann.

Oblow, E. M. (1992). Implementing Valiant's learnability theory using random sets. *Machine Learning, 8*, 45-73.

Pagallo, G. (1989). Learning DNF by decision trees. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 639-644). Detroit, Michigan: Morgan Kaufmann.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In Michalski, Carbonell & Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81-106.

Quinlan, J. R. (1987). Decision trees as probabilistic classifiers. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 31-37). Irvine, CA: Morgan Kaufmann.

Quinlan, J. R., & Rivest, R. L. (1989). Inferring decision trees using the minimum description length principle. *Information and Computation, 80*, 227-248.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning, 5*, 239-266.

Quinlan, J. R. (1991). Improved estimates for the accuracy of snmall disjuncts. *Machine Learning, 6*, 93-98.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning.* Morgan Kaufmann.

Rissanen, J. (1983). A universal prior for integers and estimation by minimum description length. *Annals of Statistics, 11*, 416-431.

Rivest, R. (1987). Learning decision lists. *Machine Learning, 2.*

Shen, Wei-Min (1992). Complementary discrimination learning with decision lists. *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 153-158). San Jose, CA: MIT Press.

Thrun, Sebastian B. (1991). *The Monk's problems: A performance comparison of different leaning algorithms.*, (CMU-CS-91-197), Pittsburgh, PA: Carnegie Mellon University, School of Computer Science.

Utgoff, P. E. (1988). Perceptron trees: A case study in hybrid concept representations. *Proceedings of the Seventh National Conference on Artificial Intelligence* (pp. 601-606). Saint Paul, MN: Morgan Kaufmann.

Utgoff, P. E. (1989). Incremental induction of decision trees. *Machine Learning, 4*, 161-186.

Utgoff, P. E. (1994). *An improved algorithm for incremental induction of decision trees*, (Technical Report 94-07), Amherst, MA: University of Massachusetts, Department of Computer Science.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM, 27*, 1134-1142.

Zheng, Z. (1993). *A benchmark for classifier learning*, (Technical Report), Sydney, Australia: University of Sydney, Basser Department of Computer Science.