Report Number: WUCS-92-26

1992-07-21

Learning k-term DNF Formulas with an Incomplete Oracle

Sally A. Goldman and H. David Mathais

We consider the problem of learning k-term DNF formulas using equivalence queries and incomplete membership queries as defined by Angluin and Slonim. We demonstrate the this model can be applied to non-monotone classes. Namely, we describe a polynomial algorithm that exactly identifies a k-term DNF formula with a k-term DNF hypothesis using incomplete membership queries and equivalence queries from the class of DNF formulas.

Learning *k*-term DNF Formulas with an Incomplete Oracle

Sally A. Goldman and H. David Mathias

WUCS-92-26

July 1992

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Learning k-term DNF Formulas with an Incomplete Membership Oracle *

Sally A. Goldman
Department of Computer Science
Washington University
St. Louis, MO 63130
sg@cs.wustl.edu

H. David Mathias
Department of Computer Science
Washington University
St. Louis, MO 63130
dmath@cs.wustl.edu

WUCS-92-26

July 21, 1992

Abstract

We consider the problem of learning k-term DNF formulas using equivalence queries and incomplete membership queries as defined by Angluin and Slonim. We demonstrate that this model can be applied to non-monotone classes. Namely, we describe a polynomial-time algorithm that exactly identifies a k-term DNF formula with a k-term DNF hypothesis using incomplete membership queries and equivalence queries from the class of DNF formulas.

*This research was supported in part by NSF grant CCR-9110108 and by a GE foundation Junior Faculty Grant. An earlier version appears in the *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, July 1992.

1 Introduction

Recently much research has been directed at understanding the importance of membership queries in obtaining efficient learning algorithms. Some concept classes known to be learnable using membership and equivalence queries are deterministic finite automata [Ang87b], read-once formulas over various bases [AHK89, HH91], read-twice DNF formulas [AP90, Han91], and Horn sentences [AFP90]. Furthermore, using Angluin's method of approximate fingerprints [Ang90], it can be shown that membership queries are necessary to efficiently learn these classes. In contrast to these classes in which membership queries are useful, Angluin and Kharitonov [AK91] have shown that, under cryptographic assumptions, membership queries do not help in learning DNF formulas (with an unbounded number of terms), read-thrice formulas, NFA's and CFG's. Recently, Aizenstein, Hellerstein and Pitt [AHP92] have shown that, assuming NP $\neq$ co-NP, no polynomial-time membership and equivalence query algorithm exists for exactly identifying read-thrice DNF formulas using hypotheses of read-thrice DNF formulas.

While membership queries nicely model an omniscient teacher who provides answers to the student's questions with perfect accuracy, in reality, even a good teacher occasionally makes errors. Thus it is important to consider the learning scenario in which the learner can ask membership queries but the answers are subject to errors. Likewise, if one views membership queries as experiments performed by the learner, then in any practical application the learner must be able to handle inconclusive results from the experiments.

Some research has been directed at understanding noise in membership queries. Sakakibara [Sak90] shows that if each membership query is erroneously answered independently at random, then an algorithm designed to work with noise-free membership queries can simply repeat each query sufficiently often so that, with high probability, the majority vote is correct. However, one would expect that if a teacher erroneously answers a question, this incorrect answer will be given whenever the question is asked. Goldman, Kearns, and Schapire [GKS90] consider a model in which the noise in the membership queries is persistent. For each instance v, when first queried, the true output of the target concept is computed and is reversed with probability p. This answer is then repeated for all future queries on v. They show that certain classes of read-once formulas are still learnable under this form of noise. Since the membership queries made by their algorithm are randomly selected in order to simulate a particular distribution, the noise can be handled by just making additional queries. While this is an appealing model, in general, it appears to be very difficult to work with.

Angluin and Slonim [AS91] introduced a model of *incomplete membership queries* in which each membership query is answered "I don't know" with probability p. Here too, this information is persistent—repeatedly making a query that was answered with "I don't know" always results in an "I don't know" answer. They show that under this model the class of monotone DNF formulas can be exactly identified with high probability. For $p \leq 1/2$, the expected running time of their algorithm is $O(m^2n^4)$.

The main result of this paper is a polynomial time algorithm to obtain exact identification in a representational sense of DNF formulas with a constant number of terms under this model of incomplete membership queries. While the equivalence queries made by our algorithm come from the representation class of general DNF formulas, with high probability the final hypothesis output will be a k-term DNF formula that is logically equivalent to the target. Given that there exist distinct but logically equivalent k-term DNF formulas, such a result is the best possible.

The learnability of DNF formulas is an important problem, and thus it is crucial to explore ways to make learning algorithms for this class robust against noise. There are several known algorithms for learning DNF formulas with a restricted number of terms using complete membership queries. Angluin [Ang88] has shown that Valiant's algorithm for learning k-CNF formulas in the PAC model can be applied to obtain a $O(n^k)$ algorithm to exactly identify k-CNF formulas using only equivalence queries. Thus one can exactly identify k-term DNF formulas in $O(n^k)$ time using the representation class of k-CNF formulas with only equivalence queries. Also, Blum and Singh [BS90] give an alternate technique to learn k-term DNF formulas in $O(n^k)$ time using only equivalence queries from the class of general DNF formulas. More recently, Blum and Rudich [BR92] have given an algorithm with $O\left(n \cdot 2^{O(k)}\right)$ expected running time for learning k-term DNF formulas using equivalence queries from the class of general DNF formulas and complete membership queries. Thus for constant k their algorithm is linear in n and it runs in polynomial time for $k = O(\lg n)$.

All of the algorithms mentioned above for learning k-term DNF formulas do *not* obtain a formula that exactly identifies the target formula in a representational sense (i.e. the final hypothesis comes from a more general representation class). In fact, Pitt and Valiant [PV88] have shown that for $k \geq 2$, the class of k-term DNF formulas cannot be exactly identified, with a k-term DNF result, in polynomial time using only equivalence queries if $P \neq NP$. In contrast to this representational hardness result, Angluin [Ang87a] has given an $O(n^{k^2})$ algorithm for learning k-term DNF formulas that constructs a k-term DNF formula logically equivalent to the target formula using equivalence and membership queries. Also, by modifying the Blum and Rudich algorithm [Blu92] one can obtain an algorithm that builds a k-term DNF formula logically equivalent to the target with an expected running time of $O(n \cdot 2^{O(k^2)})$. The key to obtaining this result is the observation that the final hypothesis output by the Blum and Rudich algorithm is a DNF formula with $O(2^{O(k)})$ terms. Thus by looking at all subsets of k terms and performing an equivalence query, one obtains a k-term formula that is logically equivalent to the target in $O(2^{O(k^2)})$ time.

The algorithm we present here achieves exact identification of k-term DNF formulas, yielding k-term DNF results, with high probability, using incomplete membership queries and equivalence queries. For $p \leq 1/2$, the expected running time is $O(n^{k^2+8})$. Observe that we must stipulate "with high probability" since it is possible that all membership queries could be answered "I don't know". By the representational hardness result of Pitt and Valiant it follows that the membership queries are needed to obtain a k-term DNF formula equivalent to the target. Thus this class provides the first non-monotone class that is learnable in polynomial time with incomplete membership queries yet cannot be learned in polynomial time without the membership queries unless $P = NP$.

2 Definitions

We begin by formally describing the model of learning from membership and equivalence queries [Ang87a]. The learner must infer an unknown *target concept* h_* chosen from some known *concept class* $\mathcal{C}$. In this paper, $\mathcal{C} = \bigcup_{n \geq 1} \mathcal{C}_n$ is parameterized by the number of variables n, and each $h \in \mathcal{C}_n$ represents a DNF formula over the instance space $\{0,1\}^n$. Also, we assume that the n variables are $x_1, x_2, \ldots, x_n$ where the value of x_i is given by the ith bit of the instance. A learning algorithm achieves *exact identification* of a concept class (with high probability) if it can infer a concept that is logically equivalent to the target concept on all inputs (with probability at least $1 - \delta$). We say an algorithm is a polynomial-time algorithm if it runs in time polynomial in n (and $\lg \frac{1}{\delta}$). In this paper we are also interested in the stronger requirement that the learner obtain a hypothesis that is equivalent to the target concept in a representational sense. That is, we want the final hypothesis to come from the same representation class as the target. We say that a learning algorithm achieves *representational exact identification* of a concept class if it outputs a hypothesis from the representational class of the target concept that achieves exact identification.

Given an instance v, if position i of v is 1 then we say x_i is true in v and $\overline{x}_i$ is false in v. Similarly, if position i of v is 0 then we say $\overline{x}_i$ is true in v and x_i is false in v. Let $h(v) = 1$ denote that formula h is true for instance v and $h(v) = 0$ denote that h is false for instance v. A *complete membership query* MQ(v) returns "yes" if $h_*(v) = 1$ and returns "no" if $h_*(v) = 0$. An *equivalence query*, Equiv(h), takes any polynomially evaluatable hypothesis h and returns "yes" if h is logically equivalent to h_* or returns a counterexample otherwise. This variation of an equivalence query in which the hypothesis need not be from the same concept class as the target concept is sometimes referred to as an *extended equivalence query*. A *positive counterexample* v is one for which $h_*(v) = 1$ but $h(v) = 0$. Likewise, a negative counterexample is one for which $h_*(v) = 0$ but $h(v) = 1$.

We now describe the model of incomplete membership queries [AS91]. An *incomplete membership oracle (IMQ)* is identical to a complete membership oracle except that it answers "I don't know" to some subset of the membership queries. The oracle determines this subset by answering "I don't know" independently with probability p the first time a membership query is made for each instance. This missing information is persistent in that all repetitions of a query result in the answer given the first time.

Finally, a k-term DNF formula h_* over n variables is a disjunction of k terms. That is, $h_* = t_1 + t_2 + \cdots + t_k$ where t_i is monomial over the n variables. We say that a variable is relevant to a particular term t_i if that variable appears in t_i, either negated or unnegated and denote the literals appearing in term t_i by $lits(t_i)$.

3 Building a Term

A key step in our algorithm is to use an example, v, for which term t of the target is positive, to generate a set of candidates for t that is "likely" to include t. More specifically, we give a technique for which the expectation is that after four positive counterexamples for t the set of candidate terms for t will include t. Then we need just distinguish t from the other candidates in the set.

As in Angluin and Slonim [AS91] we view the sample space as a lattice, with componentwise "or" and "and" as the lattice operators. The top element is the vector $\{1\}^n$ and the bottom element is the vector $\{0\}^n$. The elements are partially ordered by $\leq$, where $v \leq w$ if and only if each bit in v is less than or equal to the corresponding bit in w. The *descendants* (respectively, *ancestors*) of a vector v are all vectors w in the sample space such that $w \leq v$ (respectively, $w \geq v$). For any non-negative constant d, the *d-descendants* are all of the descendants w of v that can be obtained from v by replacing at most d 1's by 0's. The *d-ancestors* are defined analogously.

For a monotone term, by moving down in the lattice (i.e. changing a 1 to 0), the term can only be "turned off". Thus every monotone term can be uniquely represented by the minimum vector in the ordering $\leq$ for which it is true. Although we can no longer associate a term with a vector in the lattice as in the monotone case, this way of viewing the sample space is quite useful. For a term t, let $max(t)$ denote the vector obtained by turning on all literals relevant to t and setting all non-relevant bit positions to 1. Thus $max(t)$ is the maximum vector in the ordering $\leq$ that satisfies the term t. Likewise, let $min(t)$ denote the vector obtained by turning on all literals relevant to t

and setting all non-relevant bit positions to 0. Thus $min(t)$ is the minimum vector in the ordering $\leq$ that satisfies t. For example, if $n = 5$ and $t = x_1\overline{x}_3x_4$, $max(t) = 11011$ and $min(t) = 10010$. Finally, observe that t is the conjunction of literals that are true in both $min(t)$ and $max(t)$.

We now describe a simple algorithm using incomplete membership queries to construct the candidate set for a term t. Suppose we could use incomplete membership queries to reduce v, a positive counterexample, to $min(t)$ and increase v to $max(t)$. Then $min(t)$ and $max(t)$ could be combined to obtain t. To deal with the difficulty created by the incomplete membership queries we use procedures similar to the REDUCE1 procedure given by Angluin and Slonim. Namely, in trying to find $max(t)$ we query all sufficiently close ancestors of v. If a "yes" reply is given then we continue from this vector. Otherwise, if all queries are answered with "no" or "I don't know" then it is likely that $max(t)$ is either the vector for which the last "yes" reply was given or one of the vectors for which an "I don't know" answer was given. We place all such vectors in a set, *maxset*, that contains the vectors that might be $max(t)$. So the procedure, SEARCH-FOR-MAX (shown in Figure 1), adds to *maxset* a set of vectors that is "likely" to contain $max(t)$. There is a dual procedure, SEARCH-FOR-MIN, for adding to *minset* a set of vectors that is likely to contain $min(t)$. Finally, the candidate set for the term is constructed by including all terms COMBINE(x, y) for $x \in$ *minset* and $y \in$ *maxset*, where COMBINE(x, y) denotes the conjunction of literals that are true in both x and y. Observe that if $min(t) \in$ *minset* and $max(t) \in$ *maxset* then this candidate set will include t as desired. Furthermore, the following observation will be used in several proofs.

Observation 1 *All vectors v that are descendants of positive vector x and ancestors of positive vector y are classified as positive by the hypothesis consisting of the term* COMBINE(x, y).

Proof: The term COMBINE(x, y) contains *only* variables that have the same value in x and y. Thus, since v, by definition, *must* have the same value for these variables it will be classified as positive. ■

```
SEARCH-FOR-MAX $(v, d, maxset)$ :

 $maxset \leftarrow maxset \cup \{v\}$
 Let $A$ be the $d$-ancestors of $v$
 For each $a \in A$ in breadth-first order
     If $a$ not previously queried
         Then If IMQ($a$) = "I don't know"
             Then $maxset \leftarrow maxset \cup \{a\}$
         Else If IMQ($a$) = "yes"
             Then SEARCH-FOR-MAX $(a, d, maxset)$
```

Figure 1: Algorithm for updating *maxset* given a positive counterexample v.

We now argue that if d is chosen appropriately, the expectation is that after four positive counterexamples, t will be placed in its candidate set. It follows from the work of Angluin and Slonim that if no previously queried positive vectors are encountered by SEARCH-FOR-MAX then

$$\Pr[max(t) \notin maxset\,] \ \leq\ p^{2^{d+1}-2}/(1-p).$$

Solving for p yields that choosing $d = \lceil f(p) \rceil$ for

$$f(p) = \lg\left(2 + \frac{1 + \lg(1/(1-p))}{\lg(1/p)}\right) - 1 \qquad (1)$$

guarantees that $\Pr[max(t) \notin maxset] \leq \frac{1}{2}$. Observe that for $p = \frac{1}{2}$, $d = 1$ is sufficient. As shown by Angluin and Slonim, in general, selecting $d = \left\lceil \lg \frac{1}{1-p} + \lg\lg \frac{1}{1-p} \right\rceil$ suffices. A dual result holds for *minset.*

We now must argue that by calling SEARCH-FOR-MAX and SEARCH-FOR-MIN with each positive counterexample, we have a "reasonable" chance of finding either $min(t)$ or $max(t)$. We say a call to SEARCH-FOR-MAX (and respectively SEARCH-FOR-MIN) *has a clear path* if it does not encounter any previously queried positive vectors.

From Observation 1, it follows that if SEARCH-FOR-MIN and SEARCH-FOR-MAX are called with an example v for which $t(v) = 1$, yet all terms of COMBINE(*minset*, *maxset*) classify v as negative, then either SEARCH-FOR-MIN or SEARCH-FOR-MAX must have a clear path. Furthermore, if SEARCH-FOR-MAX (respectively, SEARCH-FOR-MIN) has a clear path, then it follows from the work of Angluin and Slonim that with probability at least $1/2$, $max(t)$ is added to *maxset* (respectively, $min(t)$ is added to *minset*). Thus the expectation is that, for $d \geq \lceil f(p) \rceil$, after four positive counterexamples $min(t)$ and $max(t)$ are found and thus t is included in its candidate set. By applying Chernoff bounds it is easily shown that after receiving $16 \log(1/\delta)$ positive counterexamples t will be in its candidate set with probability at least $1 - \delta$.

In order to obtain representational exact identification, we must reduce the candidates for t to a single term, which in this case must be t itself since no other term is logically equivalent to t. What types of terms, besides t itself, can be placed in the candidate set for t? One possibility is to include a term $t' \neq t$ that is subsumed by t, where t' is subsumed by t if t' logically implies t. In other words, t' is subsumed by t if $lits(t) \subset lits(t')$. This situation occurs when combining an instance from *maxset* that is a descendant of $max(t)$ with an instance from *minset* that is an ancestor of $min(t)$. Finally, the remaining terms in the candidate set are terms t' that are not subsumed by t. We say such candidates are *undesirable* with respect to t since there exists an instance v for which $t(v) = 0$ and $t'(v) = 1$. If the target concept is just the single term t then it is easily seen that such *undesirable* terms are eliminated by negative counterexamples. Furthermore, once the undesirable terms are removed from the candidate set, those terms that are subsumed by t can be eliminated efficiently .

4 Learning k-term DNF Formulas

In this section we present our main result, an algorithm to obtain representational exact identification of a k-term DNF formula, with high probability, using incomplete membership queries and equivalence queries. Although the final hypothesis output is a k-term DNF formula, the hypotheses used for the equivalence queries are general DNF formulas.

A natural application of the technique described in the previous section is to use an equivalence query to get a positive counterexample v and then construct a candidate set for some new term of the target formula. However, when using membership queries to reduce v to $min(t)$ and increase v to $max(t)$ for some new term t, it is possible that we reduce v to $min(t_1)$ and increase v to $max(t_2)$, for two different terms t_1 and t_2, that when combined do not create a new term of the target. This situation could occur because v satisfied t_1 and t_2, or it could be that these terms were turned on when moving through the lattice. To deal with this difficulty we use a discriminant as originally defined by Angluin [Ang87a]. Informally, a discriminant provides a mechanism for focusing on a single term of the target formula.

4.1 Discriminants

We now formally describe the discriminant. These definitions are taken from Angluin's paper. Let $I_k = \{(i,j) \mid 1 \leq i \neq j \leq k\}$. A *discriminant* for a k-term DNF target h_* is an indexed collection of literals L_{ij} for $(i,j) \in I_k$ such that:

1. For every $(i,j) \in I_k$, L_{ij} is a literal that is in t_i and not in t_j.

2. If t_i and t_j contain a complementary pair of literals, then L_{ij} and L_{ji} are a complementary pair of literals.

3. For each $i = 1, ..., k$, the set $\{L_{ij} : j \neq i\}$ does not contain a complementary pair.

As Angluin shows, there are $k(2n)^{k(k-1)}$ possible discriminants for a k-term DNF formula over n variables. A *valid discriminant* is one that adheres to the above rules. For each i, let L_{i*} denote the set of literals $\{L_{ij} : j \neq i\}$. Analogously, let L_{*i} denote the set of literals $\{L_{ji} : j \neq i\}$. Thus L_{i*} is a subset of the literals in t_i. It is L_{*i} that allows the discriminant to focus on a single term. Turning off all of the literals in L_{*i} accomplishes this since, by definition, L_{*i} contains a literal in every term except t_i.

A k-term DNF formula h_* is *redundant* if the formula obtained by removing any term from h_* is logically equivalent to h_*. Otherwise, it is *nonredundant*. Angluin has shown that for all nonredundant DNF formulas, a valid discriminant exists.

4.2 The Basic Framework

In this section we describe the basic framework of our algorithm. Then, in the next section, we describe how to deal with incomplete membership queries. Using complete membership queries Angluin [Ang87a] has described an algorithm for representationally exactly identifying the class of k-term DNF formulas. Her algorithm builds on the following algorithm for learning monomials. Let v be the counterexample returned when making an equivalence query with the empty hypothesis. The algorithm then makes membership queries for each instance obtained by flipping exactly one bit in v and uses the information from these queries to determine the relevant variables. In extending this algorithm to learn k-term DNF formulas, the interaction between the terms of the target creates difficulties. Angluin handles this problem by using the discriminant to find a positive counterexample that turns on a single new term t and to calculate the relevant variables of t.

As in Angluin's algorithm, we use a discriminant to modify the positive counterexample given by the equivalence query to satisfy a single new term t_i. In addition, we use this discriminant to constrain the search for $min(t_i)$ and $max(t_i)$ so that no other terms are turned on in the process.

Recall that L_{i*} is a subset of the literals that appear in t_i. If we are building t_i these literals must be true, so we constrain the search for $min(t_i)$ and $max(t_i)$ by considering only those vectors in which the literals in L_{i*} are true. While this step is not necessary, clearly it speeds up the search. More importantly, to ensure that only t_i is true, we further constrain the search by turning off all literals in L_{*i}. Recall that L_{*i} is a set of literals that for every $j \neq i$ contains some literal in t_j, but not in t_i. For vector v and literal l, let $v[l]$ denote the vector obtained by turning l off in v. Thus, $v[L_{*i}]$ is the vector v with all of the literals in L_{*i} turned off. By turning off the literals of L_{*i} we ensure that for every $j \neq i$, $t_j(v[L_{*i}]) = 0$. Therefore, if $h_*(v[L_{*i}]) = 1$ then it must be the case that $t_i(v) = 1$. So constraining the search using L_{*i} guarantees that the search never finds the MIN or MAX vectors for a term other than t_i (by wandering into the positive space of another term). Constraining the search also has the effect that we may not find $min(t_i)$ or $max(t_i)$ because we have fixed the value of some variables irrelevant to t_i. Hence, the term resulting from combining the MIN and MAX vectors obtained by this constrained search may include some variables irrelevant to t_i. Fortunately the discriminant contains enough information to determine which of the constrained variables are relevant to t_i.

Lemma 1 *Any literal l such that $\bar{l} \in L_{*i}$ but $l \notin L_{i*}$, is not relevant to t_i.*

Proof: We use a proof by contradiction. Assume that l is relevant to t_i, $\bar{l} \in L_{*i}$, yet $l \notin L_{i*}$. Since $\bar{l}$ is in L_{*i} it must be that $L_{ji} = \bar{l}$ for some $j \neq i$. But by the definition of a discriminant if $L_{ji} = \bar{l}$ and $l \in t_i$ then $L_{ij} = l$. Thus $l \in L_{i*}$, giving the desired contradiction. ■

Thus, to build the portion of the hypothesis corresponding to term i, we use a new combining function, COMBINE2(v_1, v_2) that is the conjunction of unconstrained literals that are true in both v_1 and v_2 and the constrained literals in L_{i*}.

4.3 Handling Incomplete Membership Queries

We now describe how our algorithm can be made robust against incomplete membership queries. We first consider the procedure BUILD-FORMULA that, given a valid discriminant for h_*, constructs a formula isomorphic to h_* with high probability.

As described above, we use the discriminant to constrain the searches for MIN and MAX. The procedure SEARCH-FOR-MAX2$(v, d, \mathit{maxset}(t_i))$, that builds the set $\mathit{maxset}(t_i)$ is like SEARCH-FOR-MAX except that it constrains the search for $\mathit{max}(t_i)$ by considering only those vectors in which the literals in L_{i*} are on and the literals in L_{*i} are off. The procedure SEARCH-FOR-MIN2 functions analogously.

Given positive counterexample v, with complete membership queries we can determine the terms t_i of h_* for which $t_i(v[L_{*i}]) = 1$. However, since the membership queries are incomplete we cannot always make this determination. It is possible that for some positive counterexample v and for all t_i such that $t_i(v) = 1$, the query IMQ$(v[L_{*i}]) =$ "I don't know". Therefore, in order to ensure progress, we must try to build a term t_i if IMQ$(v[L_{*i}]) =$ "yes" or "I don't know". Thus it is possible that BUILD-FORMULA will try to build from v some term t_j for which $t_j(v) = 0$. Suppose that $v' = v[L_{*j}]$

is an ancestor of $max(t_j)$ yet $\mathrm{IMQ}(v') =$ "I don't know". Then BUILD-FORMULA will try to build t_j from v' and clearly it is possible that $max(t_j)$ could be found during SEARCH-FOR-MIN2 and added to $minset(t_j)$. Since we do not repeat any membership queries on positive vectors, it is possible that $max(t_j)$ is "blocked off" by previously asked queries and thus not added to $maxset(t_j)$. In this case, t_j may not be built. For this reason we introduce $searchset(t_j) = minset(t_j) \cup maxset(t_j)$ where $1 \leq j \leq k$. The hypothesis h is then created by

$$h \leftarrow h \vee \left(\bigvee_{x,y \in searchset(t_i)} \text{COMBINE2}(x, y)\right).$$

By keeping a single set of MAX/MIN candidates for each term, we eliminate the possibility that some term is never built.

The procedure BUILD-FORMULA is shown in Figure 4.3 and a sample execution is shown in Figure 4.3. Observe that this algorithm assumes that k is known a priori. If k is not known the standard doubling trick can be applied. The algorithm LEARN-KTERM-DNF(also shown in Figure 4.3) simply calculates all possible discriminants for h_* and then calls BUILD-FORMULA for each discriminant until it returns successfully.

5 Analysis

In this section we prove that LEARN-KTERM-DNF obtains representational exact identification of the target k-term DNF formula with probability at least $1-\delta$. Furthermore, LEARN-KTERM-DNF runs in polynomial time. We first prove that if BUILD-FORMULA is called with a valid discriminant for h_*, then with probability at least $1 - \delta$ it will return a hypothesis isomorphic to h_*. We then complete the correctness proof for LEARN-KTERM-DNF by showing, with probability at least $1 - \delta$, that for some call of BUILD-FORMULA a k-term DNF formula logically equivalent to the target is obtained. Finally, we prove that the time and sample complexities are as polynomial in n and $\lg 1/\delta$.

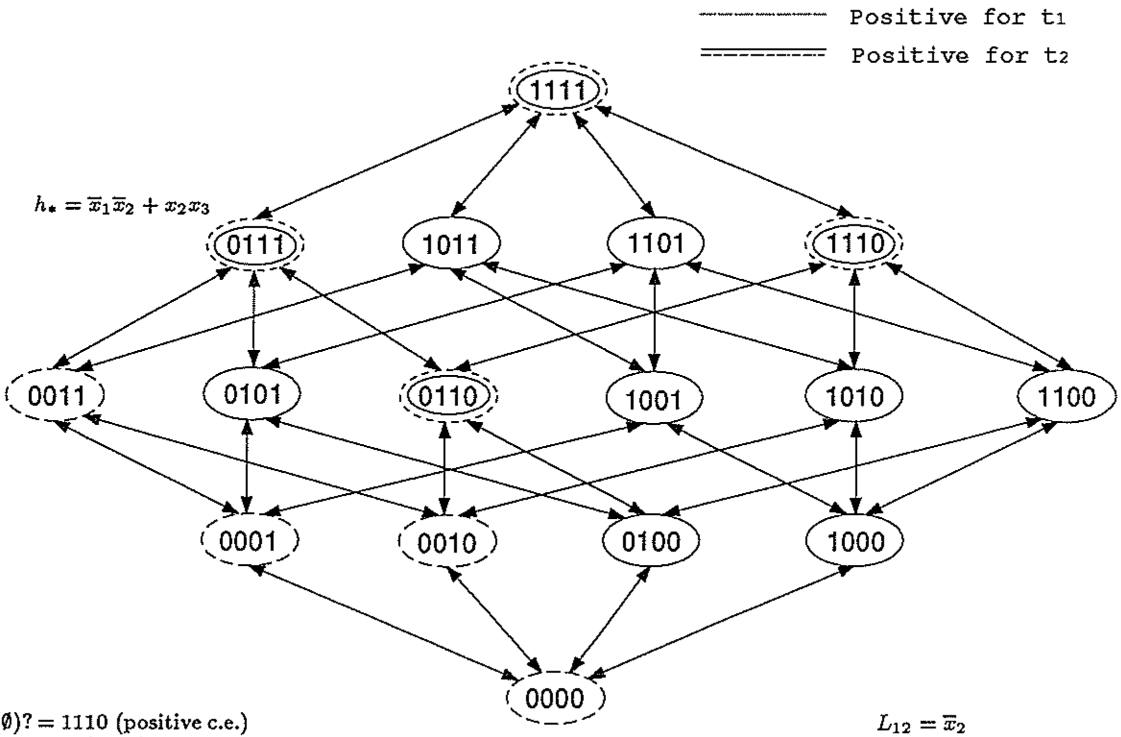
```
BUILD-FORMULA$(\{L_{ij}\}, k, d, \delta)$:

For $1 \leq i \leq k$
    $searchset(t_i) \leftarrow \emptyset$
$pos\text{-}ce \leftarrow 0$
$h \leftarrow \emptyset$
While Equiv$(h) \neq$ "yes"
    Let $v$ be the counterexample returned
    If $h(v) = 1$
        Then remove from $h$ all terms $t$ s.t. $t(v) = 1$
    Else
        $pos\text{-}ce \leftarrow pos\text{-}ce + 1$
        If $pos\text{-}ce > 16k \log(1/\delta)$ Then Return $fail$
        Use IMQ to find $I = \{i$ such that $h_*(v[L_{*i}]) =$ "yes" or "I don't know"$\}$
        If $I = \emptyset$ Then Return $fail$
        For each $i \in I$
            $searchset(t_i) \leftarrow searchset(t_i) \cup$ SEARCH-FOR-MIN2 $(v[L_{*i}], d)$
            $searchset(t_i) \leftarrow searchset(t_i) \cup$ SEARCH-FOR-MAX2 $(v[L_{*i}], d)$
            $h \leftarrow h \vee \left(\bigvee_{x,y \in searchset(t_i)} \text{COMBINE2}(x, y)\right)$
$h' \leftarrow$ CLEAN-FORMULA$(h)$
If $h'$ is a $k$-term DNF formula
    Return $h'$
    Else Return $fail$
```

```
LEARN-KTERM-DNF$(k, d, \delta)$:

    For all possible discriminants $\{L_{ij}\}$
        $h \leftarrow$ BUILD-FORMULA$(\{L_{ij}\}, k, d, \delta)$
        If $h \neq fail$ Then Return $h$
    Return $fail$
```

Figure 2: The algorithm, LEARN-KTERM-DNF, to obtain representational exact identification of h_* is shown. For each possible discriminant, it calls the procedure BUILD-FORMULA that constructs the target if given a valid discriminant, or report failure otherwise. The procedure CLEAN-FORMULA simple removes from h all terms t_j such that some term t_i in h subsumes t_j.



Equiv($\emptyset$)? = 1110 (positive c.e.) $L_{12} = \overline{x}_2$

IMQ(1010) = "I don't know"

IMQ(1110) = "yes" $L_{21} = x_2$

after SEARCH-FOR-MIN2(1010), $searchset(t_1) = \{0010, 0000\}$

after SEARCH-FOR-MAX2(1010), $searchset(t_1) = \{0010, 0000, 1011\}$

$h = \overline{x}_1\overline{x}_2x_3\overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + x_1\overline{x}_2x_3x_4 + \overline{x}_1\overline{x}_2\overline{x}_4 + \overline{x}_2x_3 + \overline{x}_2$

after SEARCH-FOR-MIN2(1110), $searchset(t_2) = \{0110, 0100\}$

after SEARCH-FOR-MAX2(1110), $searchset(t_2) = \{0110, 0100, 1111\}$

$h = \overline{x}_1\overline{x}_2x_3\overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + x_1\overline{x}_2x_3x_4 + \overline{x}_1\overline{x}_2\overline{x}_4 + \overline{x}_2x_3 + \overline{x}_2 + \overline{x}_1x_2x_3\overline{x}_4 +$
$\overline{x}_1x_2\overline{x}_3\overline{x}_4 + x_1x_2x_3x_4 + \overline{x}_1x_2\overline{x}_4 + x_2x_3 + x_2$

Equiv(h)? = 0100 (negative c.e.)

remove from h all terms t such that $t(0100) = 1$

$h = \overline{x}_1\overline{x}_2x_3\overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + x_1\overline{x}_2x_3x_4 + \overline{x}_1\overline{x}_2\overline{x}_4 + \overline{x}_2x_3 + \overline{x}_2 + \overline{x}_1x_2x_3\overline{x}_4 + x_1x_2x_3x_4 + x_2x_3$

Equiv(h)? = 1011 (negative c.e.)

remove from h all terms t such that $t(1011) = 1$

$h = \overline{x}_1\overline{x}_2x_3\overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_4 + \overline{x}_1x_2x_3\overline{x}_4 + x_1x_2x_3x_4 + x_2x_3$

Equiv(h)? = 0001 (positive c.e.)

IMQ(0001) = "yes"

IMQ(0101) = "no"

after SEARCH-FOR-MIN2(0001), $searchset(t_1) = \{0010, 0000, 1011, 1000\}$

after SEARCH-FOR-MAX2(0001), $searchset(t_1) = \{0010, 0000, 1011, 1000, 0011\}$

$h = \overline{x}_1\overline{x}_2x_3\overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_4 + \overline{x}_1x_2x_3\overline{x}_4 + x_1x_2x_3x_4 + x_2x_3 + x_1\overline{x}_2x_3x_4 + x_1\overline{x}_2\overline{x}_3\overline{x}_4 +$
$\overline{x}_1\overline{x}_2x_3x_4 + \overline{x}_2x_3 + \overline{x}_2\overline{x}_4 + \overline{x}_1\overline{x}_2x_3 + \overline{x}_2 + \overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1\overline{x}_2 + x_1\overline{x}_2 + \overline{x}_2x_3x_4$

Equiv(h)? = 1000 (negative c.e.)

remove from h all terms t such that $t(1000) = 1$

$h = \overline{x}_1\overline{x}_2x_3\overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_4 + \overline{x}_1x_2x_3\overline{x}_4 + x_1x_2x_3x_4 + x_2x_3 +$
$x_1\overline{x}_2x_3x_4 + \overline{x}_1\overline{x}_2x_3x_4 + \overline{x}_2x_3 + \overline{x}_1\overline{x}_2x_3 + \overline{x}_1\overline{x}_2 + \overline{x}_2x_3x_4$

Equiv(h)? = 1011 (negative c.e.)

remove from h all terms t such that $t(1011) = 1$

$h = \overline{x}_1\overline{x}_2x_3\overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_4 + \overline{x}_1x_2x_3\overline{x}_4 + x_1x_2x_3x_4 + x_2x_3 + \overline{x}_1\overline{x}_2x_3x_4 + \overline{x}_1\overline{x}_2x_3 + \overline{x}_1\overline{x}_2$

Equiv(h)? = "yes"

CLEAN-FORMULA(h) $= x_2x_3 + \overline{x}_1\overline{x}_2$

Figure 3: Example execution of BUILD-FORMULA with a correct discriminant. The incomplete membership. queries made by SEARCH-FOR-MIN2 and SEARCH-FOR-MAX2 are not shown.

5.1 Correctness of BUILD-FORMULA when given discriminant for target

Throughout this section we shall assume that LEARN-KTERM-DNF makes a single call to BUILD-FORMULA with a valid discriminant L for h_*. In this case, we actually prove the stronger result that with probability at least $1 - \delta$, BUILD-FORMULA outputs a hypothesis that is isomorphic to the target. We obtain this stronger result by proving that any hypothesis logically equivalent to h_* that has L as a valid discriminant is, in fact, isomorphic to the target.

Theorem 1 *Any two distinct but logically equivalent k-term DNF formulas do not have any common valid discriminant.*

Proof: We use a proof by contradiction. Let f_1 and f_2 be two non-isomorphic k-term DNF formulas that are logically equivalent. Suppose that L is a valid discriminant for both f_1 and f_2. Since f_1 and f_2 are not isomorphic there must exist some term t in f_1 that is not in f_2. Without loss of generality assume that t is the first term of f_1, $t' \neq t$ is the first term of f_2, and l is some literal in t that is not in t'. (If no such literal exists, then just reverse the roles of t and t'.) Consider an example x for which all literals in L_{*1} and l are off, and all literals in t' are on. Since all literals in L_{*1} are off it follows that $f_1(x) = t(x)$ and $f_2(x) = t'(x)$. However $t(x) = 0$ and $t'(x) = 1$. Thus f_1 and f_2 are not logically equivalent giving the desired contradiction. ■

Thus to obtain a formula isomorphic to the target, we need just find a formula with the given discriminant that is logically equivalent to the target. We proceed by proving that after receiving enough positive counterexamples, with high probability BUILD-FORMULA will create a hypothesis that contains all of the terms of h_*. To prove this result we use the following additional notation and series of lemmas.

Let $V_i^+ = \{v \mid t_i(v) = 1 \wedge \text{ all literals in } L_{*i} \text{ off}\}$. Thus V_i^+ are the positive vectors that may be queried when building term t_i. Observe that for $1 \leq i \leq k$, the sets V_i^+ are disjoint. Let v be a positive counterexample from an equivalence query made by BUILD-FORMULA. If IMQ$(v[L_{*i}])$ ="I don't know" we may try building some t_i for which $t_i(v[L_{*i}]) = 0$. This introduces two types of positive counterexamples relative to a term t_i — those for which $t_i(v) = 1$ and those for which $t_i(v) = 0$. However, observe that for every positive counterexample v, $t_i(v[L_{*i}]) = 1$ for some term t_i.

We now show that for every positive counterexample v such that $t_i(v[L_{*i}]) = 1$,

$$\Pr[\mathit{min}(t_i) \Rightarrow \mathit{searchset}(t_i) \text{ or } \mathit{max}(t_i) \Rightarrow \mathit{searchset}(t_i)] \ \geq \ \frac{1}{2}$$

where "$x \Rightarrow \mathit{set}$" is read "$x$ added to set" and denotes that

$$\mathit{set} \leftarrow \{\mathit{set} \cup \{x\} \mid \text{previously } x \notin \mathit{set}\}.$$

Let a *successful iteration* of the main loop of BUILD-FORMULA be one in which for some term t_i either $\mathit{min}(t_i) \Rightarrow \mathit{searchset}(t_i)$ or $\mathit{max}(t_i) \Rightarrow \mathit{searchset}(t_i)$. We now show that when BUILD-FORMULA receives a positive counterexample the probability of having a successful iteration is at least 1/2.

Lemma 2 *If* SEARCH-FOR-MAX2 *is called with $d \geq \lceil f(p) \rceil$ and no previously queried positive vectors are encountered then* $\Pr[\mathit{max}(t_i) \Rightarrow \mathit{searchset}(t_i)] \geq \frac{1}{2}$ *for some term t_i of h_*.*

Proof: This lemma follows immediately from the observation that for every positive counterexample v, $t_i(v[L_{*i}]) = 1$ for some new term t_i of h_*, and Lemmas 3 and 4 of Angluin and Slonim [AS91]. ■

There is a dual result for SEARCH-FOR-MIN2. In order to apply Lemma 2 it is essential that when either SEARCH-FOR-MIN2 or SEARCH-FOR-MAX2 is called, no previously queried positive vectors are encountered. That is, either SEARCH-FOR-MIN2 or SEARCH-FOR-MAX2 must have a clear path. The following series of lemmas proves that this is the case.

Lemma 3 *Let v be a vector such that $t_i(v) = 1$. If v is an ancestor of a positive vector in $\mathit{searchset}(t_i)$ and a descendant of a positive vector in $\mathit{searchset}(t_i)$ then $h(v) = 1$.*

Proof: We use a proof by contradiction. Suppose $h(v) = 0$. Let v_{min} be a minimum positive vector in $\mathit{searchset}(t_i)$ such that v is an ancestor of v_{min}. Likewise, let v_{max} be a maximum positive vector in $\mathit{searchset}(t_i)$ such that v is a descendant of v_{max}. By Observation 1, $t =$ COMBINE2(v_{min}, v_{max}) classifies all vectors that are both descendants of v_{max} and ancestors of v_{min} as positive. Furthermore, $t_i \subseteq t$ and thus t is never removed by a negative counterexample. Finally, since $t(v) = 1$ it follows that $h(v) = 1$, contradicting the initial assumption. ■

We now prove that if *max*(t_i) and *min*(t_i) are not in *searchset*(t_i) and v is a counterexample for which $t_i(v) = 1$, then either SEARCH-FOR-MIN2 or SEARCH-FOR-MAX2 has a clear path.

Lemma 4 *If max*$(t_i) \notin$ *searchset*(t_i) *and min*$(t_i) \notin$ *searchset*(t_i) *then for any counterexample* $v \in V_i^+$*, no descendants of* v *in* V_i^+ *have been queried or no ancestors of* v *in* V_i^+ *have been queried by the call to* BUILD-FORMULA *with discriminant* L*.*

Proof: Observe that the vectors in V_i^+ are only queried when building *searchset*(t_i). Assume that an ancestor of v in V_i^+ and a descendant of v in V_i^+ have been queried. Note that all positive vectors queries when trying to build t_i are placed in *searchset*(t_i), and thus *searchset*(t_i) contains both positive ancestors and descendants of v. Finally, by Lemma 3, v is classified as positive, contradicting that v is a positive counterexample. ■

Finally, in order to ensure that for every positive counterexample, with probability 1/2, either *min*(t_i) or *max*(t_i) is added to *searchset*(t_i) for some term t_i, we must show that if *min*(t_i) (respectively, *max*(t_i)) is already in *searchset*(t_i) and $t_i(v) = 1$ then SEARCH-FOR-MAX2 (respectively, SEARCH-FOR-MIN2) has a clear path.

Lemma 5 *If min*$(t_i) \in$ *searchset*(t_i) *then for any counterexample* $v \in V_i^+$*, no ancestors of* v *in* V_i^+ *have been previously queried by the call of* BUILD-FORMULA *with discriminant* L*.*

Proof: Assume that *min*$(t_i) \in$ *searchset*(t_i) and let $v' \in V_i^+$ be a maximum positive ancestor of v that has already been queried. Observe that $v' \in$ *searchset*(t_i) since *searchset*(t_i) contains all positive vectors queried when trying to build t_i. Since *min*(t_i) is a positive descendant of v and v' is a positive ancestor of v, it follows from Lemma 3 that $h(v) = 1$ contradicting that v is a positive counterexample. ■

A dual result holds for the case *max*$(t_i) \in$ *searchset*(t_i). We now prove that after receiving enough positive counterexamples, BUILD-FORMULA will have build all terms of the target with high probability.

Lemma 6 *For a non-redundant k-term DNF formula h_*, if* BUILD-FORMULA *is called with a valid discriminant for h_* then with probability at least $1 - \delta$ after receiving $16k \log(1/\delta)$ positive counterexamples the hypothesis constructed by* BUILD-FORMULA *will contain all terms of h_*.*

Proof: It follows from Lemmas 2, 3, 4, and 5 that the probability of a successful run of BUILD-FORMULA is at least $1/2$. Observe that $2k$ successful runs are needed to guarantee that all k MIN vectors and all k MAX vectors are found. Since each positive counterexample has a probability of $1/2$ of yielding a success, the expected number of positive counterexamples required is $4k$, and by Chernoff bounds the probability that more than $16k \log(1/\delta)$ positive counterexamples are obtained without yielding $2k$ successful runs is bounded by δ. Thus, with probability at least $1-\delta$, after receiving $16k \log(1/\delta)$ positive counterexamples, $\mathit{max}(t_i) \in \mathit{searchset}(t_i)$ and $\mathit{min}(t_i) \in \mathit{searchset}(t_i)$ for $1 \leq i \leq k$ and so each term of h_* will be included in the hypothesis created by BUILD-FORMULA. ■

Finally, we prove that CLEAN-FORMULA outputs a formula isomorphic the the target. Recall that there are only three types of terms in the hypothesis, those that appear in h_*, those that are subsumed by one of the terms that appear in h_*, and those that are undesirable with respect to some term of h_*. Since CLEAN-FORMULA just removes the terms of the hypothesis that are directly subsumed by some other term in the hypothesis, clearly the resulting formula will be logically equivalent to the target. So we need just prove that all undesirable terms are eliminated by negative counterexamples prior to the call to CLEAN-FORMULA.

Lemma 7 *For any term t' that is in the candidate set for term t_i and is undesirable with respect to t_i, there exists an instance v for which $t'(v) = 1$, yet $h_*(v) = 0$.*

Proof: Since t' is undesirable with respect to t_i, there exists an instance v' such that $t'(v') = 1$ and $t_i(v') = 0$. Consider the instance $v = v'[L_{*i}]$. Since SEARCH-FOR-MAX2 and SEARCH-FOR-MIN2 are constrained so that all literals in L_{*i} are off it is not hard to show that $t'(v) = 1$ and $t_i(v) = 0$. Finally, since L_{*i} contains a literal from all terms t_j for $j \neq i$ it follows that $t_j(v) = 0$, and thus $h_*(v) = 0$. ■

From the above lemma it immediately follows that when BUILD-FORMULA achieves exact identification, all undesirable terms have been removed by negative counterexamples. The only other types of terms are those from h_* and those subsumed by a term in h_*, and thus we can complete the proof that BUILD-FORMULA outputs a formula that is isomorphic to the target.

Theorem 2 *For a non-redundant k-term DNF formula h_*, if* LEARN-KTERM-DNF *calls* BUILD-FORMULA *a single time with a valid discriminant for h_* then, with probability at least $1 - \delta$,* BUILD-FORMULA *outputs a hypothesis that is isomorphic to h_*.*

Proof: It follows from Lemma 6 that with probability at least $1 - \delta$, at some point during the execution of BUILD-FORMULA, the hypothesis created will contain all terms in the target formula. Furthermore, after possibly some negative counterexamples, the formula constructed by BUILD-FORMULA will be logically equivalent to the target, and thus by Lemma 7, CLEAN-FORMULA needs only to remove terms t_j that are subsumed by t_i. Since CLEAN-FORMULA evaluates all pairs of terms t_j, t_k in the hypothesis and removes those terms t_j such that $lits(t_j) \supset lits(t_k)$, with probability at least $1 - \delta$, LEARN-KTERM-DNF returns a k-term DNF formula that is isomorphic to h_*. ■

5.2 Correctness of LEARN-KTERM-DNF

In this section we prove that, with high probability, LEARN-KTERM-DNF outputs a k-term DNF formula logically equivalent to the target. We have shown that it succeeds when calling BUILD-FORMULA with a valid discriminant for the target. Thus we need just consider what could happen when other potential discriminants are given to BUILD-FORMULA.

Theorem 3 *For a non-redundant k-term DNF formula h_*,* LEARN-KTERM-DNF*, obtains representational exact identification of h_* with probability at least $1 - \delta$.*

Proof: The procedure BUILD-FORMULA may be called with a discriminant that falls into one of three categories: a discriminant that is valid for h_*, a discriminant that is valid for some k-term DNF formula f that is logically equivalent to h_*, or an invalid discriminant. Observe that the lemmas used for proving Theorem 2 assume that

BUILD-FORMULA has not been previously called with a different potential discriminant. What happens if BUILD-FORMULA is called (and fails) on some invalid discriminants before being called with a valid discriminant? Note that the analysis relies on the independence of "I don't know" answers only when BUILD-FORMULA is called with a valid discriminant. Thus we can handle this technicality by assuming that the answers for incomplete membership queries are generated by an oracle that first performs the coin flips for the incomplete membership queries made by BUILD-FORMULA when given the first valid discriminant and uses these answers for the other runs of BUILD-FORMULA. Thus, from Theorem 2 we can conclude that when BUILD-FORMULA is called with a valid discriminant for h_*, then with probability at least $1-\delta$ it outputs a k-term DNF formula that is isomorphic to h_*.

What happens when BUILD-FORMULA is given an invalid discriminant? We guarantee that it does not run indefinitely by limiting the number of positive counterexamples seen in any call of BUILD-FORMULA to $16k \log(1/\delta)$. Note that it is possible for BUILD-FORMULA to learn the target when given an invalid discriminant. However, this occurs with low probability and does not affect the analysis.

The other possibility we must contend with is the situation in which BUILD-FORMULA is called with a valid discriminant for some k-term DNF formula f that is logically equivalent to h_* before it is called with a valid discriminant for h_*. When this occurs, by the same argument as in Theorem 2, with high probability LEARN-KTERM-DNF will return f, achieving representational exact identification. ■

Observe that if we are provided with an additional oracle that given a hypothesis returns "yes" if the hypothesis is isomorphic to the target, then by simply querying this oracle with each result from CLEAN-FORMULA we can continue until we finally call BUILD-FORMULA with a valid discriminant for h_*. Thus if provided with this additional oracle LEARN-KTERM-DNF can be modified to output a formula isomorphic to the target with probability at least $1-\delta$.

5.3 Time and Sample Complexity

In this section we prove that LEARN-KTERM-DNF runs in time polynomial in n and $\log(1/\delta)$ for k and p constant.

Theorem 4 *For a non-redundant k-term DNF formula h_*,* LEARN-KTERM-DNF *uses*

- $O\left(n^{k^2+2d+2}\log^3(1/\delta)\right)$ *equivalence queries,*
- $O\left(n^{k^2+d+1}\log(1/\delta)\right)$ *incomplete membership queries and*
- $O\left(n^{k^2+4d+4}\log^6(1/\delta)\right)$ *time*

where $d = \lceil f(p)\rceil$[1].

Proof: We first bound the number of incomplete membership and equivalence queries used. Angluin [Ang87a] showed that there are $k(2n)^{k(k-1)}$ possible discriminants to consider. Thus, we need just multiply this quantity by the number of equivalence and incomplete membership queries made by each call to BUILD-FORMULA.

Recall that at most $16k\log(1/\delta)$ positive counterexamples are obtained in a given call to BUILD-FORMULA. Now for each positive counterexample v, k incomplete membership queries are used to check if $h_*(v[L_{*i}])$ = "yes" or "I don't know". In addition, at most $O(n^{d+1})$ incomplete membership queries are made in the search for $min(t_i)$ and $max(t_i)$ for $1 \le i \le k$. Thus $O(k^2n^{d+1}\log(1/\delta))$ incomplete membership queries are made during a single call to BUILD-FORMULA. Since $O(n^{d+1})$ incomplete membership queries are made during the searches for $min(t_i)$ and $max(t_i)$, $O(n^{d+1})$ vectors are added to $searchset(t_i)$ for each positive counterexample. This results in

$$\sum_{i=1}^{16k\log(1/\delta)} (O(i\cdot n^{(d+1)}))^2 = O\left(k^3n^{2(d+1)}\log^3(1/\delta)\right)$$

terms being added to the hypothesis for each term of h_*. Since every negative counterexample removes at least one of these terms, $O\left(k^4n^{2(d+1)}\log^3(1/\delta)\right)$ negative counterexamples can occur during a single call of BUILD-FORMULA. Finally, the time complexity follows from observing that $O(k)$ time is spent for each positive counterexample,

[1] Recall that Angluin and Slonim have shown that $f(p) \le \lg\frac{1}{1-p} + \lg\lg\frac{1}{1-p}$ and thus selecting $d = \left\lceil \lg\frac{1}{1-p} + \lg\lg\frac{1}{1-p}\right\rceil$ suffices.

$O(k^4 n^{2(d+1)} \log^3(1/\delta))$ time is spent for each negative counterexample since all terms in the hypothesis are examined, and $O(1)$ time is spent for each incomplete membership query. ■

6 Concluding Remarks

We have given an efficient algorithm that with high probability exactly identifies, in a representational sense, any k-term DNF formula using equivalence queries and incomplete membership queries. Like the algorithm given by Angluin and Slonim [AS91], LEARN-KTERM-DNF can be modified to handle persistent false negative errors in answers to the membership queries. In this model all negative instances are correctly answered by the membership oracle but each positive instance, with probability p, is answered "no" by the first membership query made for it. The key observations here are that all queries answered "yes" are reliable, and that Theorems 3 and 4 hold even if all "no" queries were answered with "I don't know". Thus we need just modify SEARCH-FOR-MIN, SEARCH-FOR-MAX, and BUILD-FORMULA to treat all "no" answers as if they were "I don't know" answers.

One interesting open question is to determine if the class of DNF formulas with more than a constant number of terms can be exactly identified (even in just a logical sense) using equivalence queries and incomplete membership queries. In particular, it may be possible to use the techniques of Blum and Rudich [BR92] to obtain such a result. More generally, it would be interesting to determine whether some of the other classes that are known to be learnable using equivalence queries and complete membership queries are still efficiently learnable under the model of incomplete membership queries. Finally, other models of noise in membership queries should be explored.

Acknowledgements

We thank Rose Gamble, Rob Schapire and Ellen Witte for helpful discussions and for their comments on a preliminary version of this paper. We are also grateful to the COLT program committee for their comments.

References

[AP90] H. Aizenstein and L. Pitt. Exact learning of read-twice DNF formulas. In *Proceedings of the Thirty Second Annual Symposium on Foundations of Computer Science*, pages 170–179, October 1991.

[AHP92] H. Aizenstein, L. Hellerstein, and L. Pitt. Read-thrice DNF is hard to learn with membership and equivalence queries. To appear in *Proceedings of the Thirty Third Annual Symposium on Foundations of Computer Science*, October 1992.

[Ang87a] D. Angluin. Learning k-term DNF formulas using queries and counterexamples. Technical Report YALEU/DCS/RR-559, Yale University, August 1987.

[Ang87b] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, November 1987.

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[Ang90] D. Angluin. Negative results for equivalence queries. *Machine Learning*, 5:121–150, 1990.

[AFP90] D. Angluin, M. Frazier, and L. Pitt. Learning conjunctions of horn clauses. In *Proceedings of the Thirty First Annual Symposium on Foundations of Computer Science*, pages 186–192, October 1990.

[AHK89] D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. Technical Report UCB/CSD 89/528, University of California Berkeley Computer Science Division, 1989.

[AK91] D. Angluin and M. Kharitonov. When won't membership queries help? In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, pages 444–454, May 1991.

[AS91] D. Angluin and D. Slonim. Learning monotone DNF with an incomplete membership oracle. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 139–146, August 1991.

[Blu92] A. Blum. Personal Communication.

[BR92] A. Blum and S. Rudich. Fast learning of k-term DNF formulas with queries. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, pages 382–389, May 1992.

[BS90] A. Blum and M. Singh. Learning functions of k terms. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 144–153, August 1990.

[GKS90] S. Goldman, M. Kearns, and R. Schapire. Exact identification of circuits using fixed points of amplification functions. In *Proceedings of the Thirty First Annual Symposium on Foundations of Computer Science*, pages 193–202, October 1990.

[Han91] T. Hancock. Learning 2μ DNF formulas and $k\mu$ decision trees. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 199–209, August 1991.

[HH91] T. Hancock and L. Hellerstein. Learning read-once formulas over fields and extended bases. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 326–336, August 1991.

[PV88] L. Pitt and L. Valiant. Computational limitations on learning from examples. *Journal of the Association for Computing Machinery*, 35(4):965–984, 1988.

[Sak90] Y. Sakakibara. On learning from queries and counterexamples in the presence of noise. *Information Processing Letters.* To appear.