In *Proc. Seventh Annual ACM Conference on Computational Learning Theory*, p. 237–245, 1994. Warning: missing six few references—fixed in proceedings.

Learning with Queries but Incomplete Information (Extended Abstract)

Robert H. Sloan*
Dept. of Electrical Eng. and Computer Science
University of Illinois at Chicago
Chicago, IL 60607
sloan@eecs.uic.edu

György Turán§
Dept. of Math., Stat., and Comp. Sci.
University of Illinois at Chicago,
Automata Theory Research Group
Hungarian Academy of Sciences, Szeged
U11557@uicvm.uic.edu

Abstract

We investigate learning with membership and equivalence queries assuming that the information provided to the learner is incomplete. By incomplete we mean that some of the membership queries may be answered by "I don't know." This model is a worst-case version of the incomplete membership query model of Angluin and Slonim. It attempts to model practical learning situations, including an experiment of Lang and Baum that we describe, where the teacher may be unable to answer reliably some queries that are critical for the learning algorithm.

We present algorithms to learn monotone k-term DNF with membership queries only, and to learn monotone DNF with membership and equivalence queries. Compared to the complete information case, the query complexity increases by an additive term linear in the number of "I don't know" answers received. We also observe that the blowup in the number of queries can in general be exponential for both our new model and the incomplete membership model.

1 Introduction

Recently, many researchers have investigated learning from membership and equivalence queries. Algorithms have been found for deterministic finite automata, monotone DNF formulas, k-term DNF formulas, various read-once formulas, Horn formulas, and Boolean decision trees, among other classes [2–5, **?**–6, 9, **?**].

Intuitively, many learning algorithms that use membership and equivalence queries work as follows: Some number of equivalence queries are used to find positive instances in the "heart" of the concept, and membership queries are used to trace out the "boundary" of the concept.

In the past several years, the study of the power of membership queries has been extended to cover the effects of noise and incomplete information on membership queries [**?**, 10, 11, 14]. These papers have assumed either that the noisy responses are randomly generated, or that if a single instance is repeatedly queried, then the correct classification will eventually be given with high probability. In Sakakibara's model [14], each membership query is erroneously answered independently at random, and one can defeat the noise by querying points many times and taking a majority vote. This method also works for both of Bultman's models [10]. In the *incomplete membership query* model [**?**], all queries of the same instance give the same response, so repeated queries of an instance do not help. In this model, the response to a membership query is either correct or "I don't know." Angluin and Slonim introduced this model and obtained strong positive results in it; subsequently Goldman and Mathias obtained even stronger positive results [**?**, 11].

Given our intuitive understanding of membership queries, we believe the reason that strong positive results can be obtained for the incomplete membership model is that with random "Don't knows," one can still trace out the boundary of the concept, since it is very unlikely that all the "Don't knows" would pile up right on the boundary.

In this paper, we study the *limited membership query* model, which is similar to the incomplete membership query model, but does not have random generation of the "Don't knows." Instead, we consider the case where there is a fixed number ℓ of "Don't knows," and study the *worst case* over all possible sets of ℓ "Don't knows." The learner does not know the parameter ℓ; rather we study its performance as a function of ℓ.

We study the worst case because we believe that in practice "Don't know" responses will often be given for precisely those instances where typical algorithms ask membership queries. Presumably in applications the "membership oracle" is a human domain expert, and there are many cognitive psychology studies showing that people are typically very inconsistent in deciding where the precise boundary of a

*Partially supported by NSF grant CCR-9108753.

§Partially supported by NSF grant CCR-9208170, Esprit BRA ILP, Project 6020, and OTKA 501.

concept lies. (See, e.g. [1].)

Exactly this difficulty arose when two researchers tried to apply some computational learning theory algorithms involving membership queries to a real-world problem. Baum developed an algorithm for learning neural nets from examples and membership queries [8], and with Lang attempted to apply it to the problem of recognizing hand-written digits [12]. This experiment was a failure because the the membership questions the algorithm posed were too difficult for people to answer reliably. The algorithm typically asked membership queries on, say, a random-looking blur midway between a "5" and a "7," and the humans being queried gave very inconsistent responses [12].

We believe there will often be a complicated dependence of the "Don't knows" on the target concept, and this is what we seek to model. We do this by looking for learning algorithms that work for *any* set of "Don't knows" as long as this set is not too large. Angluin and Slonim [?] mention finding such a model as a topic for further research in their article on incomplete membership queries.

Recently, Angluin and Kriķis [?, ?] independently formulated a model that is quite similar to the one we consider in this paper. They assumed that a fixed number of membership queries is answered *incorrectly* (as opposed to being answered by "Don't knows"). In other words, learning algorithms in their model are required to tolerate a bounded number of *lies*. Answers to equivalence queries are assumed to be correct. Some comparisons between these two models are given below in Sections 2, 5, and 6. In addition, the malicious misclassification noise model for PAC learning [15] is also related to these models.

In the next section we give some definitions; in Section 3, we give a key subroutine for finding one term of a monotone DNF starting from a given positive instance. Our main results are presented in Section 4. We present algorithms for learning monotone k-term DNF from limited membership queries alone, and another algorithm for learning arbitrary monotone DNF from limited membership and equivalence queries. The algorithm for monotone k-term DNF is the more novel and complex of the two. In Section 5 we compare learning in several models related to the one presented in this paper. We also give a general meta-algorithm to convert any membership and equivalence query algorithm to use limited membership queries at the cost of an exponential blowup in the number of queries used, and show that at least for some classes such a blowup is required.

2 Definitions

We follow the standard model of learning from membership and equivalence queries [4]. The goal of the learner is to infer an unknown *target concept* from some given *concept class* $\mathcal{C}$ over a given *instance space* or *domain*. We will view concepts interchangeably as subsets of the instance space and as 0-1 functions on the instance space. For most of this paper we will view the concept class as being $\mathcal{C} = \bigcup_{n\geq 1} \mathcal{C}_n$ where $\mathcal{C}_n$ is a class of sets of $\{0,1\}^n$. We also require the running time of our learning algorithms to be polynomial in the natural parameter(s). For all the learning algorithms presented in this paper, polynomial running time follows from the arguments proving the query bounds, and it will not be discussed separately in what follows.

For the standard noise-free model, we require *proper exact identification*. This means that the learner must output precisely the correct target concept, and that the output must be in a specified syntactic form.

Fix a target concept c. An ordinary or *complete membership query* on instance v, denoted MQ(v), returns $c(v)$. In this paper we study the *limited membership oracle*. This oracle returns either the correct classification of the instance, $c(v)$, or "I don't know," which we will denote by $\perp$. Queries to the limited membership oracle about instance v will be denoted LMQ(v). In what follows, we often use expressions such as "v has LMQ 1" for LMQ(v) $= 1$. We assume that the limited membership oracle fixes the set of instances on which it replies $\perp$ in advance, and we consider the worst-case performance of our learning algorithms over all possible sets of $\perp$ instances of a given size.

This limited membership oracle is of less help to the learner than the *incomplete membership oracle*, which has a similar behavior but picks the subset of instances to be labeled $\perp$ randomly [?]. As a result, one expects to be able to tolerate many fewer $\perp$ responses. With the incomplete membership oracle, one can sometimes learn even if the probability of a $\perp$ response is a constant independent of n. We, on the other hand, will be looking at tolerating $\perp$ responses for only up to a polynomial number of the instances in $\{0,1\}^n$.

Note that in the remainder of this paper, a *positive instance* means any instance v in the target concept regardless of whether LMQ(v) $= 1$ or LMQ(v) $= \perp$, and similarly for negative instances.

In addition to membership queries, we will sometimes also have *equivalence queries*, which take as input a hypothesis concept h from the learner and return either "correct" or a *counterexample*. A counterexample is an instance v such that the target concept and h classify v differently.

We now must consider what constitutes correct identification, and how the LMQ oracle and the equivalence oracle interact. Whenever we are studying learning from limited membership queries alone, we will consider identification to be correct when the learner outputs a hypothesis that agrees with the target concept on all instances, not counting those instances with LMQ $\perp$. In our standard model for learning with limited membership and equivalence queries, we assume that instances classified $\perp$ by the limited membership oracle cannot be given as counterexamples by the equivalence oracle, and that correct identification is achieved once the learner outputs a concept which agrees with the target concept on all instances not counting those instances with LMQ $\perp$. This corresponds to our view that $\perp$ instances represent borderline instances whose classification is irrelevant or meaningless.

In the *strict* model, we require the learner to output a concept that agrees with the target concept on all instances. In this case, we assume that the equivalence oracle can give any instance whatsoever as a counterexample.

In the *malicious membership query* model [**?**], membership queries are answered by a *malicious membership oracle* that may give incorrect answers to membership queries. The counterexamples provided for the equivalence queries are assumed to be correct, and thus they may be used to "catch" previous lies. As "Don't knows" can be replaced by, say, the response "No," a limited membership oracle can be used to simulate a malicious membership oracle. Thus positive results in the malicious membership model are stronger, in that they imply positive results in our model.

Since this paper will mostly be concerned with learning monotone DNF, let us introduce some terminology used in that learning problem. As do others (e.g. [**?**, 11]), we will view the instance space as a partially ordered set. The top element is the vector 1^n and the bottom element is 0^n. The *descendants* of an instance v are all instances w such that $w \leq v$, where $x \leq y$ if each bit of x is less than or equal to the corresponding bit of y. The *ancestors* of an element v are all elements w such that $w \geq v$. The *children* of v are all descendants of v with Hamming distance 1 from v; the *parents* of v are all ancestors of v with Hamming distance 1 from v.

For convenience, we consider every instance to be synonymous with the monotone term containing exactly those unnegated variables that correspond to 1's in the instance, and we will use the words "instance" and "term" interchangeably. By a term, we always mean a monotone term. Notice that a term is satisfied by exactly its ancestors. In the other direction, we will say that term t *covers* instance v if and only if v satisfies t.

Finally, for a formula f in DNF, we will use $|f|$ to refer to the number of terms in f.

3 A key subroutine for monotone DNF: delimiting a term

In this section we describe an operation that is used repeatedly in our algorithms for learning monotone DNF. Algorithm DELIMIT takes a positive instance and finds a set of candidates for a term in the target concept covering the instance. This algorithm plays the role of the algorithms called "Reduce" in other works on learning monotone DNF [**?**, **?**]. We choose a different name because those algorithms output a single monomial, whereas Algorithm DELIMIT finds a set inside of which a term must lie.

Consider, for example, the situation in Figure 1 where the target concept is x_1x_2, and we start with the known positive instance $p = 1111$. With complete information, we would begin by querying each child of p, updating p to be the first positive child found. This process would be repeated until eventually we had $p = 1100$. After determining by membership queries that every child of p is negative, we could stop.

Because Algorithm DELIMIT can make only limited membership queries, it may encounter the difficulty that some positive instance p has children that are all "Don't know," or a mix of "No" and "Don't know." For instance, in the example in Figure 1, all queries of children of 1101 return 0 or $\perp$. In this case, Algorithm DELIMIT (whose detailed code is given in Figure 2), continues by querying all the children of *all* the "Don't know" instances. Should it ever get another 1 response to a limited membership query, it replaces p by that instance.

Eventually the algorithm reaches a point where p is a positive instance, and there is a set DK of descendants of p (possibly empty) such that the limited membership oracle responded $\perp$ for every instance in DK, and all other descendants of p are known to be negative instances. We call this instance p a *root*. Since p is a positive instance, and we know that every descendant of p not in set *DK* is negative, a term of the monotone DNF must lie someplace in the set $DK \cup \{p\}$. This set is outlined in Figure 3.

What we have described so far is what we call the "down" pass of Algorithm DELIMIT; it is the portion of the algorithm described in Figure 2 from the beginning until the label "Up:". (Ignore for now the variable *OffBits*.) The rest of Algorithm DELIMIT attempts to thin this set of possible terms, and also gathers other useful information.

Any instance corresponding to a term of the target concept must have only positive ancestors. In the "up" pass of Algorithm DELIMIT, we first ensure that no instance in *DK* has any ancestor with LMQ 0, and then finally output a thinned set of possible terms T. On the example of Figures 1 and 3, 0100 is deleted from *DK* because LMQ(0101) $=$ 0. If LMQ(1010) $=$ 0, then 1000 will also be deleted. If, in addition, LMQ(1110) $=$ 0 holds as well, then 0100, 1000, and 1100 will all be deleted.

We summarize our discussion in the following lemma.

Lemma 1 *Let T and P be the outputs of running Algorithm* DELIMIT *on positive instance p when the target concept is a monotone DNF. Then*

1. *T contains a term of the target concept covering p.*
2. *For every $t \in T$, every instance covered by t has* LMQ *1 or $\perp$.*
3. *For any instance v such that* LMQ$(v) = 1$*, and for any term $t \in T$ that covers v, term t covers some instance in P that is a descendant of v.*

Proof sketch: We argued above that T must contain a term of the target concept, except that we ignored the variable *OffBits* which is used in the "down" phase to improve efficiency. If we query the child q of known positive instance p and receive a 0 response, then we know that switching off the bit position that distinguishes q from p in any descendant of p will lead to a negative instance. Indeed, this new instance will be a descendant of q. The variable *OffBits* keeps track of those bit positions that must be 1, allowing Algorithm DELIMIT to save some queries.

For 2, we know that *root* is a positive instance, so all the ancestors of *root* are positive too. For every other $t \in T$ the

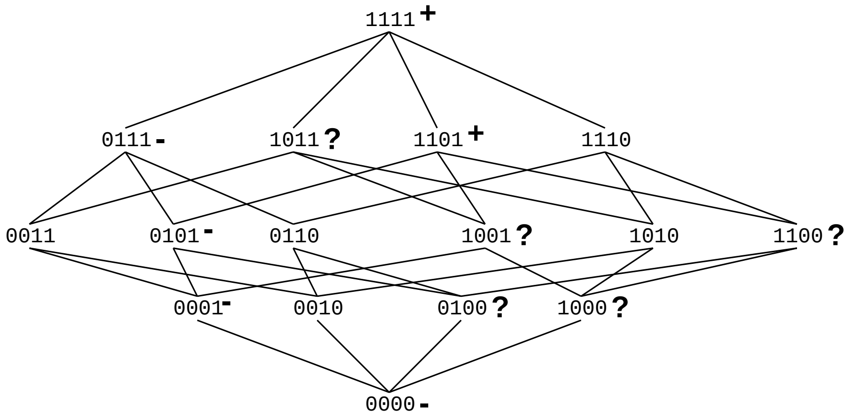

Figure 1: Example run of Algorithm DELIMIT for target concept x_1x_2. Boldface $+$'s, $-$'s and ?s indicate responses 1, 0, and $\perp$, respectively, of the LMQ oracle.

DELIMIT(p) /* p a positive instance */
Outputs: Set T of possible terms, set P of instances with LMQ 1.

for $1 \leq j \leq n$
  OffBits$[j] \leftarrow 0$
root $\leftarrow p$
Down: $C \leftarrow$ all children of *root* that are $\geq$ *OffBits*
$DK \leftarrow \emptyset$
while $C \neq \emptyset$
  Delete some maximal element a from C
  case of LMQ(a)
  1: *root* $\leftarrow a$
    goto Down
  0: **if** a is immediate child of *root*
      $j \leftarrow$ position on which a and *root* differ
      OffBits$[j] \leftarrow 1$
  $\perp$: $DK \leftarrow DK \cup \{a\}$
    $C \leftarrow C \cup \{x : x$ is a child of a and $x \geq$ *OffBits*$\}$
Up: $P \leftarrow \{$*root*$\}$
for each $a \in DK$
  $A(a) \leftarrow$ parents(a)
  for each $b \in A(a)$
    case of LMQ(b)
    1: $P \leftarrow P \cup \{b\}$
    0: Delete all descendants of b from DK.
      break (innermost) for loop
    $\perp$: $A(a) \leftarrow A(a) \cup$ parents(b)
$T \leftarrow DK \cup \{$*root*$\}$

Figure 2: Algorithm DELIMIT to find candidates for a monotone term covering instance p.

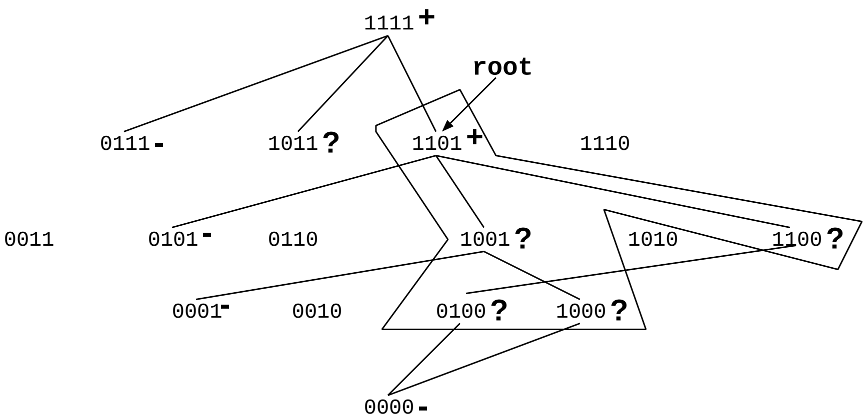

Figure 3: The set of candidate terms obtained at the end of the "down" phase of Algorithm DELIMIT.

queries made in the "up" phase of the algorithm guarantee that the ancestors of t have LMQ 1 or $\perp$.

As *root* is covered by all terms in T, so are all its ancestors. For other positive instances covered by some $t \neq$ *root* $\in T$, the set P contains all the minimal (proper) ancestors of t with LMQ 1, so part 3 of the lemma is correct as well. □

Lemma 2 *Algorithm* DELIMIT *makes at most $n\ell + n$ limited membership queries, where ℓ is the number of $\perp$ responses received. In general, any sequence of s calls to Algorithm* DELIMIT *for the same target concept makes at most $n(\ell + s)$ limited membership queries.*

Proof sketch: Every instance queried is either the child of the original root or a previously queried instance with LMQ 1, or else is Hamming distance 1 from a $\perp$ instance.

Each time we receive a $\perp$ response on an instance v, we could in principle need to query all the children of v in the "down" phase of the algorithm, and then all the parents of v in the "up" phase of the algorithm, except that there is a parent or a child of v that we must have queried before querying v. This leads to at most $n - 1$ queries for each $\perp$ received, plus 1 for the $\perp$ query itself. As long as we "remember" the answers to all queries, this holds for any number of calls to Algorithm DELIMIT on the same concept.

Every time we query the child of the original root or a previously queried instance with LMQ 1, and get a 1 or 0 response, the quantity "number of 1's in *root* minus number of 1's in *OffBits*" must decrease by 1, so there can be at most n queries of this sort, in each call of DELIMIT. □

4 Results for monotone DNF

4.1 Monotone monomials and k-term DNF from membership alone

We begin with a very simple application of Algorithm DELIMIT to learn monotone monomials from limited membership queries. This should elucidate the basic ideas at the heart of the more complicated algorithm for learning arbitrary monotone k-term DNF from limited membership queries that follows.

Theorem 1 *Monotone monomials can be learned from no more than $n\ell + n + 1$ limited membership queries, where ℓ is the number of $\perp$ responses received.*

Proof sketch: The method is to run Algorithm DELIMIT starting with the all 1's vector, and then output the monomial m that is the intersection of all the terms in set P. That is, the learner's output will have a variable if and only if that variable is in every term in P. (If either LMQ$(11\cdots1) = 0$, or LMQ$(11\cdots1) = \perp$ and the "down" phase of Algorithm DELIMIT finds no positive instances, then we will output FALSE.)

Monomial m covers every instance in P, so by Lemma 1-3, it must cover all instances v such that LMQ$(v) = 1$. On the other hand, since m is the intersection of positive instances, it must either be the correct monotone monomial or an ancestor of the correct monotone monomial, so m cannot cover any negative instance.

Lemma 2 counts all queries made by this algorithm except the first query to $11\cdots1$. □

We have a lower bound for this problem as well.

Theorem 2 *For any integer c, if the limited membership oracle gives $\sum_{k=0}^{c}\binom{n}{k}$ responses of $\perp$, then any learner can be forced to use at least $\binom{n}{c+1} - 1$ limited membership queries* not counting those answered $\perp$ *to learn monotone monomials.*

Proof A target concept defined by a monomial of length $n - (c+1)$ contains no instance with more than $(c+1)$ 0's, and it contains exactly one instance with $(c+1)$ 0's. Now let us assume that all queries of a learning algorithm on instances containing c or fewer 0's are answered by $\perp$, and its first $\binom{n}{c+1} - 2$ queries of instances with at least $(c+1)$ 0's are answered by "No" (0). After that there are at least two unqueried instances with $(c+1)$ 0's. The corresponding two monomials are both consistent and they differ on instances that did not receive a $\perp$ response. Hence the learning algorithm needs at least one more query. □

Corollary 3 *For any fixed constant c, if the limited membership oracle gives $O(n^c)$ responses of $\perp$, then the learner can be forced to use $\Omega\left(n^{c+1}\right)$ limited membership queries to learn monotone monomials.* □

Now in Figure 4, we present Algorithm ONLYLMQ to learn monotone k-term DNF from limited membership queries alone. Algorithm ONLYLMQ initializes its hypothesis h to be the empty formula, and repeats the following:

By exhaustive search, an instance a with LMQ$(a) = 1$ that is not covered by hypothesis h is found. If none is found, then h is correct and is output. (This is the portion of Algorithm ONLYLMQ above the blank line.)

Now the instance a is given to Algorithm DELIMIT, which returns the sets T and P. If there is a single term in T that covers all the positive instances in P not already covered by h, then this term is added to h and we go back to the line labeled "term" to begin the main loop again.

Otherwise, we initialize *CandidateTerms* to be T and initialize *Pos* to be those instances in P that are not covered by h. We call Algorithm DELIMIT on every instance in *Pos*, and for each call add the instances from T to *CandidateTerms* and the instances from P not covered by h to *Pos*. Then we try to cover the newly enlarged set *Pos* with any $j = 2$ terms from *CandidateTerms* . If we succeed, we put these terms in h; if we fail we keep enlarging *CandidateTerms* , *Pos*, and j in the same manner and trying to cover all instances in *Pos* with j terms from *CandidateTerms* .

Theorem 4 *Algorithm* ONLYLMQ *learns monotone k-term DNF from $O\left(k(n^k + n^2\ell)\right)$ limited membership queries, where ℓ is the number of $\perp$ responses it receives.*

Proof sketch: All the terms added to the hypothesis are at some point in a set T output by Algorithm DELIMIT. Therefore, by Lemma 1-2, no term that we ever add to our hypothesis ever errs by classifying an instance with LMQ 0 as positive.

Furthermore, since Lemma 1-1 guarantees that each time we call Algorithm DELIMIT from an instance we get at least one term covering that instance, each time Algorithm ONLYLMQ is in the loop at line "try" it must eventually succeed in covering the set *Pos* with some number of terms. Since there are only a finite number of positive instances, this means that the algorithm eventually terminates with a hypothesis that correctly classifies all non-$\perp$ instances.

Thus the only thing we need to show for the correctness of the algorithm is that the final hypothesis of Algorithm ONLYLMQ contains at most k terms. The following lemmas provide the gist of that argument.

Lemma 3 *Whenever Algorithm* ONLYLMQ *attempts to cover the set of positive instances Pos with a disjunction of j terms from CandidateTerms at the line labeled "try," the set CandidateTerms contains at least j distinct terms of the target concept. Furthermore, none of those j terms of the target concept is implied by the current hypothesis h.*

Proof sketch for Lemma 3: The proof is by induction on j. The base case is provided by Lemma 1-1, which says that there must must be a term of the target concept in T at the end of the first call to Algorithm DELIMIT. Since this term covers the instance on which Algorithm DELIMIT was called, it must be a new term.

If at line "try" we are trying to cover *Pos* with $j+1$ terms, then we know that we tried and failed to cover the previous version of *Pos* with j terms. By the inductive hypothesis, this implies that the previous version of *CandidateTerms* contained at least j terms of the target concept. If the previous version of *CandidateTerms* contained more than j terms of the target concept, then we are done. Otherwise, it contained j terms of the target concept. Hence there was some instance in the previous version of *Pos* not covered by any of those terms, and a term of the target concept covering this instance must have been added to the set *CandidateTerms* by the final round of calls to Algorithm DELIMIT. □

Lemma 4 *Let h' be the hypothesis that Algorithm* ONLYLMQ *obtains when it updates its hypothesis h. Then h' contains all instances with LMQ 1 that are covered by any term in CandidateTerms .*

Proof sketch for Lemma 4: Assume the contrary holds for h, *CandidateTerms* , and h'. Let $t \in$ *CandidateTerms* be a term and x be an instance with LMQ 1 such that such that $h'(x) = 0$ and $t(x) = 1$.

Term t was added to *CandidateTerms* by some call to Algorithm DELIMIT. At that call, according to Lemma 1-3, the set P must have contained an instance p such that t covered p and p is a descendant of x. Since $h'(x) = 0$, we know $h(x) = 0$, so $h(p) = 0$. Therefore, after the call to Algorithm DELIMIT when the outputs had $t \in T$ and $p \in P$, instance p was added to *Pos*. Thus since h' is satisfied by all instances in *Pos*, it must be that $h'(p) = 1$ and thus $h'(x) = 1$, a contradiction. □

Lemma 5 *Consider the hypothesis h of Algorithm* ONLYLMQ *at any point in the running of that algorithm for monotone DNF target concept c. There is a monotone DNF d consisting of at least $|h|$ terms from c such that*

$$\{v : d(v) = \text{LMQ}(v) = 1\} \subseteq \{v : h(v) = 1\}.$$

Proof sketch for Lemma 5: The proof is by induction on the number times that h is enlarged by adding new terms. We use Lemmas 3 and 4. □

Lemma 5 implies that the hypothesis h of Algorithm ONLYLMQ never contains more than k terms, which is what we needed to show.

A k-term DNF can have at most n^k maximal negative instances. This implies that each time S is initialized, it can have at most n^k elements. Also, S is initialized at most k times, as each time this happens at least one term is added to h. The bound for the number of queries then follows from Lemma 2. □

ONLYLMQ()
$h \leftarrow$ "the empty formula"
term: $S \leftarrow$ all maximal elements of $\{v \in \{0,1\}^n : h(v) = 0\}$
    do for each previously unqueried $a \in S$
        if LMQ$(a) = \perp$
            $S \leftarrow S \cup \{$children of $a\}$
    until LMQ$(a) = 1$ or all of S queried
    if all of S was queried **then** halt and **output** h

    $T, P \leftarrow$ DELIMIT(a)
    CandidateTerms $\leftarrow T$
    Pos $\leftarrow \{v \in P : h(v) = 0\}$
    for $j = 1, 2, 3, \ldots$
try: **if** a disjunction of j terms from *CandidateTerms* covers all instances in *Pos*
        then
            Add that disjunction of terms to h
            goto term
        else
            for every $p \in$ *Pos* not previously delimited (if any)
                $T, P \leftarrow$ DELIMIT(p)
                CandidateTerms $\leftarrow$ *CandidateTerms* $\cup T$
                Pos $\leftarrow$ *Pos* $\cup \{v \in P : h(v) = 0\}$

Figure 4: Algorithm for learning monotone k-term DNF from limited membership queries.

$h \leftarrow$ "the empty formula"
while $(v =$ equivalence-query$(h)) \neq$ "correct"
    if v is a negative counterexample
    then remove all terms satisfied by v from h
    else $h \leftarrow h \cup$ terms T of DELIMIT(v).
output concept h

Figure 5: Algorithm for learning monotone DNF from equivalence and limited membership queries.

4.2 Monotone DNF from membership and equivalence

Theorem 5 *Monotone DNF formulas can be learned from $n(m+\ell)$ limited membership queries together with $m+1$ equivalence queries in our standard model, or together with $m+\ell+1$ equivalence queries in the strict model, where ℓ is the number of $\perp$ responses received, and m is the number of terms in the target monotone DNF.*

Proof sketch: We modify Angluin's algorithm for learning monotone DNF from ordinary membership and equivalence queries [4], by using Algorithm DELIMIT. Our algorithm is specified in Figure 5. Note that the same algorithm works for both the standard and strict models.

The "up" pass of Algorithm DELIMIT guarantees that all terms we put in h are either correct, or err only by classifying negative instances with LMQ $\perp$ as positive. Lemma 1 guarantees that each time we call Algorithm DELIMIT we get a new term of the target monotone DNF.

In the standard model, possible "negative $\perp$ becomes positive" errors will not matter, so we will have one call to Algorithm DELIMIT and one equivalence query per term. In the strict model, we may need one extra equivalence query to eliminate each such term from the hypothesis. This implies the bounds for the number of equivalence queries. The bound for the number of LMQ's then follows from Lemma 2. □

Remark: Polynomial learnability of monotone DNF from equivalence and limited membership queries is implied by the stronger result of Angluin and Kriķis [**?**]. It is at least as difficult to learn from their malicious membership oracle as it is from our limited membership oracle, so their algorithm for monotone DNF could be used here. Our direct algorithm does, however, give somewhat better bounds on the number of queries required.

5 General bounds

In this section we discuss the relationship among the two models presented in this paper, error-free learning with equivalence and membership queries, and the model of Angluin and Kriķis.

5.1 Learning with and without "Don't knows"

We briefly describe a method for converting any algorithm for exact identification from membership and equivalence queries to one that works for limited membership queries. We can also show that in some cases an exponential blowup in the number of queries is necessary.

Theorem 6 *a) Every concept class that is learnable with m equivalence and membership queries is learnable with $2^\ell(m-\ell+1)+\ell-1$ equivalence and limited membership queries, where ℓ is the number of $\perp$ responses received.*

b) There is a concept class learnable with m equivalence and membership queries that requires $2^\ell(m-\ell+1)-1$ equivalence and limited membership queries, where ℓ is the number of $\perp$ responses received.

Proof sketch: a) Let Algorithm A exactly identify concept class $\mathcal{C}$ from at most m equivalence and membership queries. We construct a learning algorithm for equivalence and limited membership queries as follows: For each instance x such that $\mathrm{LMQ}(x)=\perp$, start running two copies of A in parallel, one assuming x is positive and one assuming x is negative. The bound follows by noticing that the simulation terminates in at most m phases. Notice that each copy of A (except one) may need to make one extra equivalence query at the end to determine whether it has the correct answer.

b) We construct a concept class $\mathcal{C}$ that is a variant of ADDRESSING [13]. Let the instance space be $X=\bigcup_{i=0}^{2^\ell} X_i$, where the X_i's are disjoint, $X_0=\{1,\ldots,\ell\}$, and $|X_i|=m-l+1$ for $1\le i\le 2^\ell$. Since $|X_0|=\ell$, each of its subsets can be viewed as an ℓ-bit number. A set $c\subseteq X$ is in $\mathcal{C}$ if and only if it has the following form. It contains exactly one element x that is not in X_0, and if i denotes the number represented by $c\cap X_0$, then that x is in X_i.

Concept class $\mathcal{C}$ can be learned by ℓ membership queries for the elements in X_0 followed by $m-\ell$ membership queries for the elements of X_i, where i is the number represented by the responses obtained in the first phase.

On the other hand, the following adversary strategy shows that at least $2^\ell(m-\ell+1)-1$ equivalence and limited membership queries are required to learn $\mathcal{C}$. The limited membership oracle responds $\perp$ to all instances in X_0. Membership queries for other elements are answered by "No." Equivalence queries are answered by providing a negative counterexample outside X_0. □

Remark: Taking $\ell=m$ in part (b) gives us the original concept class ADDRESSING, and an example where m $\perp$ responses increase the number of queries required from m for ordinary membership queries to 2^m-1 for limited membership and equivalence queries.

Notice that ADDRESSING also causes the incomplete membership query model to have an expected exponential blowup over ordinary membership queries when the probability of a $\perp$ response is a constant. For constant probability p of $\perp$, the expected number of instances in X_0 answered $\perp$ is pm. This will increase the number of queries required from m for ordinary membership queries to 2^{pm} for incomplete membership and equivalence queries.

If instead of equivalence queries one allows equivalence queries with *any set*, then such a blowup cannot occur. This follows from a result of Auer and Long [7] showing that in this model membership queries can speed up learning by only a constant factor.

5.2 The strict model

The contents of this subsection are due to Dana Angluin and Mārtiņš Kriķis.

For the purposes of this subsection, we refer to the main model discussed in this paper as the *non-strict* model, as opposed to the *strict* model. Every learning algorithm that works in the strict model can be run in the non-strict model without increasing its complexity. A relationship in the other direction can be established by a method similar to the one used in the proof of Theorem 3 in Angluin and Kriķis [**?**].

Every learning algorithm that works in the non-strict model can be adapted to work in the strict model as follows. First note that the problem that may occur when running a non-strict algorithm in the strict model is that it may receive as a counterexample to an equivalence query an instance that was previously classified as a "Don't know" in a membership query. In this case, the execution of the algorithm is interrupted. The algorithm is then restarted, remembering the instance and its classification for possible later use in answering a membership query. Since each interruption corresponds to a new "Don't know," this simulation essentially adds a multiplicative factor of ℓ to the complexity of the learning algorithm.

Hence, from the point of view of polynomial learnability, the strict and non-strict models are equivalent.

5.3 Lies versus ignorance

As noted in Section 2 and in Angluin and Kriķis [**?**], learning with malicious membership queries (MMQ's) and equivalence queries is at least as difficult as with LMQ's and equivalence queries. In order to provide an example where learning with MMQ's and equivalence queries is in fact more difficult, let us again consider the concept class ADDRESSING described above. Then the concept class can be learned with at most $m+2^\ell$ LMQ's and equivalence queries for any ℓ. On the other hand, the following adversary strategy shows that at least $\sum_{i=0}^{\ell}\binom{m}{i}-1$ MMQ's and equivalence queries are necessary. Answer "No" for all MMQ's, and provide a negative counterexample from outside X_0 for every equivalence query.

This example, with $\ell=\log m$, shows that the number of MMQ's and equivalence queries necessary to learn a concept class cannot be bounded by any polynomial in the number of LMQ's and equivalence queries. On the other hand, since in this example the number of LMQ's and equivalence queries is exponential in ℓ, it does not answer Problem (2) from Angluin and Kriķis [**?**], "Are there any concept classes polynomially learnable from equivalence queries and LMQ's that are not polynomially learnable from equivalence queries and MMQ's?"

6 Open problems

As noted in the introduction, there are many learning problems that are efficiently learnable with membership and equivalence queries. In addition to the class of monotone DNF considered in this paper and in [?], polynomial learnability from equivalence and limited membership queries follows from the general results of Angluin and Kriķis for DFAs and for Boolean decision trees [?]. Recently Angluin [?] showed that read-once DNF formulas are polynomially learnable by a randomized algorithm using MMQ's and equivalence queries. It would be interesting to consider the complexity of the other problems as well. Also, in the model of PAC learning with membership queries, it would be interesting to see whether Baum's algorithm [8] can be modified to tolerate "I don't know" answers.

Another open problem related to the results of this paper is to prove lower bounds for learning monotone DNF with both equivalence queries and LMQ's.

Acknowledgments

Robert Sloan would like to thank Sally Goldman for some conversations on this subject several summers ago.

Both authors would like to thank Dana Angluin and Mārtiņš Kriķis for the material presented in Section 5.2.

References

[1] J. R. Anderson. *Cognitive Psychology and Its Implications*. W. H. Freeman and Company, 1980.

[2] D. Angluin. Learning k-term DNF formulas using queries and counterexamples. Technical Report YALEU/DCS/RR-559, Yale University Department of Computer Science, Aug. 1987.

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

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

[5] D. Angluin, M. Frazier, and L. Pitt. Learning conjunctions of Horn clauses. *Machine Learning*, 9:147–164, 1992.

[6] D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. *J. ACM*, 40:185–210, 1993.

[7] P. Auer and P. M. Long. Simulating access to hidden information while learning. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, 1994. To appear.

[8] E. B. Baum. Neural net algorithms that learn in polynomial time from examples and queries. *IEEE Transactions on Neural Networks*, 2(1):5–19, Jan. 1991.

[9] N. Bshouty, L. Hellerstein, and T. Hancock. Learning boolean read-once formulas over generalized bases. *J. ACM*. A preliminary version appeared in *Proc. 5th Annu. Workshop on Comput. Learning Theory*, under the title "Learning Boolean read-once formulas with arbitrary symmetric and constant fan-in gates".

[10] W. J. Bultman. *Topics in the Theory of Machine Learning and Neural Computing*. PhD thesis, University of Illinois at Chicago Mathematics Department, 1991.

[11] S. A. Goldman and H. D. Mathias. Learning k-term DNF formulas with an incomplete membership oracle. In *Proc. 5th Annu. Workshop on Comput. Learning Theory*, pages 77–84. ACM Press, New York, NY, 1992.

[12] K. J. Lang and E. B. Baum. Query learning can work poorly when a human oracle is used. In *International Joint Conference on Neural Networks*, Beijing, 1992.

[13] W. Maass and G. Turán. Lower bound methods and separation results for on-line learning models. *Machine Learning*, 9:107–145, 1992.

[14] Y. Sakakibara. On learning from queries and counterexamples in the presence of noise. *Information Processing Letters*, 37(5):279–284, Mar. 1991.

[15] R. Sloan. Types of noise in data for concept learning. In *Proc. 1st Annu. Workshop on Comput. Learning Theory*, pages 91–96, San Mateo, CA, 1988. Morgan Kaufmann.