Machine Learning, 25, 237–263 (1996)
© 1996 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

On the Limits of Proper Learnability of Subclasses of DNF Formulas*

KRISHNAN PILLAIPAKKAMNATT AND VIJAY RAGHAVAN raghavan@vuse.vanderbilt.edu
Department of Computer Science
Vanderbilt University
Nashville, TN 37235

Editor: Thomas Hancock

Abstract. Bshouty, Goldman, Hancock and Matar have shown that up to $\sqrt{\log n}$ term DNF formulas can be properly learned in the exact model with equivalence and membership queries. Given standard complexity-theoretical assumptions, we show that this positive result for proper learning cannot be significantly improved in the exact model or the PAC model extended to allow membership queries. Our negative results are derived from two general techniques for proving such results in the exact model and the extended PAC model. As a further application of these techniques, we consider read-thrice DNF formulas. Here we improve on Aizenstein, Hellerstein, and Pitt's negative result for proper learning in the exact model in two ways. First, we show that their assumption of NP $\neq$ co-NP can be replaced with the weaker assumption of P $\neq$ NP. Second, we show that read-thrice DNF formulas are not properly learnable in the extended PAC model, assuming RP $\neq$ NP.

Keywords: Models of learning–exact, PAC; proper learning

1. Introduction

1.1. The DNF problem

DNF formulas are representations of Boolean functions in Disjunctive Normal Form. The question of whether DNF formulas are efficiently learnable is a central open problem in learning theory. In Valiant's model of probably approximately correct (PAC) learning (Valiant, 1984), some partial results have been obtained. For example, Pitt and Valiant (1988) have shown that for any $k \geq 2$, k-term DNF formulas are not PAC learnable as k-term DNF formulas, unless RP = NP. On the other hand, it has been shown that for any constant k, k-term DNF formulas are learnable as k-CNF formulas (Valiant, 1984), and k-term DNF formulas are learnable as k-term DNF formulas when the learner is also allowed to ask membership queries (Bshouty, et al., 1993, Berggren, 1993). But none of these results answer the question "Can arbitrary DNF formulas be learned with polynomially larger DNF hypotheses or if membership queries are allowed in the learning process?"

Angluin and Kharitonov (1991) consider whether DNF formulas can even be predicted, i.e., PAC-learned with efficiently computable hypotheses from *any* class. They give a result which can be interpreted as positive or negative, depending on one's point of view: they show that (if certain cryptographic assumptions hold) DNF formulas can either be predicted without membership queries or cannot predicted even if membership queries are allowed.

* This work was supported by an NSF grant, CCR-9212011

That is, membership queries essentially do not help in predicting DNF formulas under arbitrary distributions. The best positive result known for specific distributions is Jackson's membership-query based algorithm (Jackson, 1994) for predicting DNF formulas under the uniform distribution.

Angluin's model of exact learning (Angluin, 1988) places more stringent requirements on learning than the PAC model; therefore negative results should be easier to obtain in the exact model. Nevertheless, the question of whether DNF formulas can be exactly learned using equivalence and membership queries is open. It is easy to see that membership queries alone will not suffice; using purely information-theoretical arguments, Angluin developed the technique of "approximate fingerprints" (Angluin, 1990) to show that equivalence queries alone will not suffice either.

1.2. Subclasses of DNF formulas

Recognizing the hardness of settling the question of learnability of general DNF formulas in either of these models, researchers have studied the learnability of some subclasses of DNF formulas, placing restrictions on the class of hypotheses available to the learner. The classes of monomials, monotone DNF formulas, Horn sentences, k-DNF formulas, read-k-sat-j DNF formulas, read-twice DNF formulas, and k-term DNF formulas ((Angluin, 1988, Valiant, 1984, Angluin, Hellerstein & Karpinski, 1993, Aizenstein & Pitt, 1991, Aizenstein & Pitt, 1992, Hancock, 1991, Hancock, 1992, Berggren, 1993, Pillaipakkamnatt & Raghavan, 1995)) have all been shown to be learnable exactly and efficiently with equivalence and membership queries. Moreover, all of these classes can be learned *properly*, i.e., using only hypotheses that come from the target class of the unknown concept to be learned. Of these classes, monomials and k-DNF formulas can be properly learned with equivalence queries alone; the others provably need *both* equivalence and membership queries for efficient proper learning.

On the negative side, Aizenstein, Hellerstein, and Pitt (1992) have shown (assuming NP $\neq$ co-NP) that read-thrice DNF formulas are not exactly learnable as read-thrice DNF formulas, given both equivalence and membership queries. The requirement that equivalence queries can use only read-thrice DNF hypotheses is crucial to this result (see (Aizenstein, Hellerstein & Pitt, 1992) for an excellent discussion of such representation dependent results). If the hypotheses may come from the larger class of DNF formulas, then exactly learning read-thrice DNF formulas is just as hard as learning general DNF formulas (Angluin & Kharitonov, 1991).

An interesting question that arises out of the proper learnability of k-term DNF formulas is the following: is there a limiting number of terms k^* (where k^* could be a function of n, the number of variables) such that DNF formulas with fewer than k^* terms can be properly learned in the PAC or exact sense, but DNF formulas with more than k^* terms cannot be learned? It seems unlikely that the limiting number k^* will be obtainable in any precise sense. Thus, a reasonable line of enquiry is to establish upper and lower bounds on k^*.

On the positive side, Bshouty, Goldman, Hancock, and Matar (1993) have shown that up to $\sqrt{\log n}$ term DNF formulas can be properly learned with equivalence and membership queries. (Berggren's linear time algorithm (Berggren, 1993) for learning k-term DNF formulas translates to a weaker lower bound on k^* of $\Omega(\frac{\log\log n}{\log\log\log n})$. Blum and Rudich (1992) have shown that $O(\log n)$ term DNF formulas are exactly learnable with equivalence and membership queries, but their algorithm does not appear to be transformable to a proper learning algorithm). So the question is, can we hope to properly learn k-term DNF formulas for k "significantly" larger than $\sqrt{\log n}$? We use complexity-theoretical arguments to show that the answer to this question is "No."

1.3. Our results

We build on the technique of Aizenstein, Hellerstein, and Pitt (1992) to prove the following negative results:

1. If $P \neq NP$, then n^{α}-term DNF formulas cannot be properly and exactly learned with equivalence and membership queries, for any fixed constant $\alpha > 0$.
2. If NP is not contained in DTIME($n^{O(\log n)}$), then $2^{\sqrt{\log n}}$-term DNF formulas over n variables cannot be exactly and properly learned with equivalence and membership queries.
3. If there exists some constant $\epsilon > 0$ such that NP is not contained in DTIME($2^{n^{\epsilon}}$), there exists a constant $c > 0$ such that DNF formulas with more than $\log^c n$ terms cannot be properly and exactly learned with equivalence and membership queries.

These negative results use progressively stronger assumptions to get better upper bounds on the limit of proper learnability of DNF formulas. Specifically, the last negative result shows that the lower bound of $\sqrt{\log n}$ terms is the best possible up to a constant exponent.

Next, we address proper PAC-learnability with membership queries. Here we generalize the method used by Pitt and Valiant (1988) and Kearns, Li, Pitt, and Valiant (1987) to get negative results. This method relies on the assumption that $RP \neq NP$ and uses the difficulty of finding hypotheses from a concept class C consistent with carefully constructed sample data to prove that C is hard to learn in the PAC sense. It has been observed (Aizenstein, 1993) that there are technical difficulties in generalizing the method to get negative results for PAC-learning when membership queries are allowed; roughly, the problem is that answering membership queries on points outside of the sample data used in the method destroys the reduction involved. Nevertheless, we show that the technique of Aizenstein, Hellerstein, and Pitt can be fruitfully combined with the $RP \neq NP$ technique to address PAC-learnability with membership queries. We use this new combined technique to show that (if $RP \neq NP$) n-term DNF formulas cannot be properly PAC-learned even if membership queries are allowed.

As another application of the techniques for proving negative results, we consider read-thrice DNF formulas, the class considered by Aizenstein, Hellerstein, and Pitt. First, we show that (unless $P = NP$) read-thrice-DNF formulas cannot be properly learned with equivalence and membership queries. This is an improvement on the earlier hardness result which depends on the stronger assumption that $NP \neq$ co-NP. Second, we show that (if RP

$\neq$ NP) then read-thrice DNF formulas cannot be properly learned in the PAC model with membership queries. This last result is not directly deducible from the earlier result.

We note here that our techniques for obtaining negative results can be extended in a relatively straightforward manner to accommodate other kinds of queries. With such an extension, one can sharpen the results given above. Specifically, all the non-learnability results for k-term DNF formulas in the exact and PAC models hold if the learner is allowed all of subset, superset, equivalence, and membership queries; the negative results for read-thrice DNF formulas hold if the learner is allowed subset, equivalence, and membership queries. Out of space considerations, here we develop our techniques for negative results only for the combination of equivalence and membership queries. The interested reader is referred to (Pillaipakkamnatt, 1995) for extensions to other queries. These negative results for proper learning with an extended set of queries are to be contrasted with the positive ones in Bshouty, Cleve, Kannan, and Tamon's (1994) recent paper. They show that every concept class can be exactly (but not necessarily properly) learned by a randomized algorithm which uses subset and superset queries. In essence, non-proper subset and superset queries allow the learner to simulate an NP-oracle, thus overcoming the fundamental computational obstacle used in our techniques for negative results.

The rest of the paper is organized as follows. Section 2 contains definitions relevant to the rest of the paper. Definitions specific to the remaining sections are developed later. Section 3 is an outline of general techniques that can be used to show hardness results. The remaining sections are applications of the general techniques—section 4 contains hardness results for proper exact learning of DNF formulas; section 5 contains the corresponding results in the PAC model. Finally, section 6 contains the results for read-thrice DNF formulas.

2. Definitions and Terminology

2.1. *Preliminaries*

Let V be a set of Boolean variables. An assignment β is a mapping $V \rightarrow \{\mathit{True}, \mathit{False}\}$. The set of $2^{|V|}$ possible assignments (instances, vectors) is called the *instance space* on V. A Boolean concept c on V is a subset of the instance space. It is convenient to view such a concept as a Boolean function in the most natural way: $c(\beta) = \mathit{True}$ if and only if β is an instance of the concept.

Let $C = \cup_{n \geq 1} C_n$ be a class of Boolean concepts, where each C_n, $n \geq 1$ is a set of concepts defined over a set V_n of n Boolean variables. A *representation* $\mathcal{R}$ for the class C is a 3-tuple $\langle \pi, R, \rho \rangle$ in which:

1. π is the alphabet of the representation.

2. $R \subseteq \pi^*$ is the set of strings on π that represent concepts in C.

3. $\rho : R \rightarrow C$ is a surjective (onto) function that maps strings in R to concepts in C, i.e., for each $c \in C$, there exists at least one string h in R which satisfies $\rho(h) = c$.

Frequently, the representation itself defines the concept class. Thus, for example, the "class of k-term DNF formulas" should be interpreted as the set of Boolean functions that can be represented as DNF formulas with no more than k terms.

2.2. *Models of learning*

Angluin models the learning process using a *minimally adequate teacher* (Angluin, 1988) in the following way: The teacher picks some concept $c \in \mathcal{C}$. An exact learner or learning algorithm $\mathcal{A}$ works with a representation $\mathcal{R} = \langle \pi, R, \rho \rangle$ for a class $\mathcal{H}$, where $\mathcal{H} \supseteq \mathcal{C}$. The goal of the learner $\mathcal{A}$ is to output a string $h \in R$ such that $\rho(h) = c$. That is, the concept c is the unknown *target concept* that $\mathcal{A}$ attempts to learn: if $\mathcal{A}$ is a true learning algorithm for $\mathcal{C}$, it will eventually output such a string, regardless of which concept c is picked from $\mathcal{C}$. The class of concepts $\mathcal{H}$ used by $\mathcal{A}$ is called its *hypothesis* class. The algorithm $\mathcal{A}$ may acquire information about the target concept c by asking the teacher two kinds of queries—*equivalence queries* and *membership queries.*

An equivalence query, EQ(h), where $h \in R$, effectively asks, "Is $\rho(h) = c$?" If $\rho(h) = c$, the teacher answers "Yes"; otherwise, the teacher answers "No" and gives a *counterexample* β such that $c(\beta) \neq \rho(h)(\beta)$.

A membership query, MQ(β), effectively asks if the assignment β is an instance of the unknown concept c; the teacher answers "Yes" if $c(\beta) = \mathit{True}$ and "No" otherwise.

We say that the class $\mathcal{C} = \bigcup_{n \geq 1} \mathcal{C}_n$ of concepts is *(efficiently) learnable in the exact model in representation $\mathcal{R} = \langle \pi, R, \rho \rangle$ for the hypothesis class $\mathcal{H} \supseteq \mathcal{C}$* if there exists a learning algorithm $\mathcal{A}_{\mathcal{C},\mathcal{R}}$ and a 2-variable polynomial $p()$ such that for any unknown target concept $c \in \mathcal{C}_n$:

1. $\mathcal{A}_{\mathcal{C},\mathcal{R}}$ uses membership queries and equivalence queries of the form EQ(h), where $h \in R$, and

2. If l is the length of the shortest representation of c in $\mathcal{R}$ (i.e., $l = \min \{|h| : h \in R \text{ and } \rho(h) = c\}$), then $\mathcal{A}_{\mathcal{C},\mathcal{R}}$ uses at most $p(l, n)$ time and outputs a string $h \in \mathcal{R}$ such that $\rho(h) = c$.

In Valiant's model of learning (Valiant, 1984), the learning algorithm is required to achieve only approximate identification. Let $c \in \mathcal{C}_n$ be the unknown concept to be learned. In the *extended model* of probably, approximately, correct (PAC) learning, a learning algorithm $\mathcal{A}$ obtains information about the unknown concept c from two sources: membership queries and *sample queries.*

The response to a sample query EX() is a random *example* $\langle \beta, b \rangle$ of the unknown concept c, where β is an assignment chosen randomly according to some fixed, but arbitrary probability distribution D over the instance space and $b = c(\beta)$.

We say that the class $\mathcal{C} = \cup_{n \geq 1} \mathcal{C}_n$ of concepts is *learnable in the extended PAC model in the representation $\mathcal{R} = \langle \pi, R, \rho \rangle$ for hypothesis class $\mathcal{H} \supseteq \mathcal{C}$* if there exists an algorithm $\mathcal{A}_{\mathcal{C},\mathcal{R}}$, and a 4-variable polynomial $p()$ such that for any unknown target concept $c \in \mathcal{C}_n$ and for any distribution D over the instance space, $\mathcal{A}_{\mathcal{C},\mathcal{R}}$ takes as input an error parameter $\epsilon, 0 < \epsilon \leq 1$, and a confidence parameter $\delta, 0 < \delta \leq 1$, and satisfies the following:

1. $\mathcal{A}_{\mathcal{C},\mathcal{R}}$ uses only membership and sample queries, and
2. $\mathcal{A}_{\mathcal{C},\mathcal{R}}$ outputs a string $h \in \mathcal{R}$ such that $\text{Prob}(\sum_{\beta:c(\beta)\neq\rho(h)(\beta)} D(\beta) > \epsilon) < \delta$, and
3. If l is length of the shortest representation of c in R, $\mathcal{A}_{\mathcal{C},\mathcal{R}}$ uses at most $p(l, n, \frac{1}{\epsilon}, \frac{1}{\delta})$ total time.

For both exact and PAC-learning, we assume that the time complexity of a learning algorithm $\mathcal{A}$ is measured in a uniform cost RAM model augmented to allow for queries in the following way. Membership and sample queries are each charged a single unit of time and an equivalence query EQ(h) costs $|h|$ units of time. In both models, the cost of outputting the final hypothesis h must be accounted for in the running time of the algorithm.

2.3. *Proper learning*

In this paper, we are interested in representation-dependent or *proper* learning. That is, we have a "natural" representation $\mathcal{R} = \langle \pi, R, \rho \rangle$ which defines a target class $\mathcal{C}$ and we are concerned with whether $\mathcal{C}$ can be efficiently learned in the representation $\mathcal{R}$ for the hypothesis class $\mathcal{H} = \mathcal{C}$ in either the exact or PAC model.

As an example of a class that is properly learnable, consider the class of read-once Boolean formulas (Angluin, Hellerstein & Karpinski, 1993), i.e., the class of Boolean functions that can be represented as Boolean formulas in which each variable occurs at most once. The statement "the class of read-once Boolean formulas is properly learnable in the exact model (with a minimally adequate teacher)," means that there exists a polynomial time exact learning algorithm that uses membership queries and equivalence queries only of the form EQ(h), where h is a string representing a read-once Boolean formula.

We consider the proper learnability of read-thrice DNF formulas in this paper. A *read-thrice* DNF formula is a DNF formula in which each Boolean variable occurs at most three times.

We also consider the proper learnability of DNF formulas that have at most m terms. Here we are interested in enforcing the constraint that the hypotheses used by the learning algorithm never have more terms than the unknown DNF formula to be learned. The natural way to do this becomes clear when we examine the following equivalent definition of normal learnability of DNF formulas in the exact model.

Let $\mathcal{D}_{m,n}$ be the set of DNF formulas with at most m terms defined over the set $V_n = \{v_1, v_2, \ldots, v_n\}$ of n Boolean variables. Let $R_{m,n}$ be the set of strings in the conventional representation of $\mathcal{D}_{m,n}$ (which uses some alphabet of n characters to represent the variables in V_n, the symbol + to denote disjunction etc.) Assume that the two Boolean functions that evaluate all assignments to *True* and all assignments to *False* are represented in $R_{m,n}$ as the special 0-term DNF formulas T and F respectively. The class $\mathcal{D} = \cup_{n\geq 1} \cup_{m \geq 0} \mathcal{D}_{m,n}$ is *learnable in the exact model* if there exists an algorithm $\mathcal{A}_{\mathcal{D}}$ and two polynomial functions $p()$ and $q()$ such that for each $c \in \mathcal{D}_{m,n}$:

1. $\mathcal{A}_{\mathcal{D}}$ uses only membership queries and equivalence queries of the form EQ(h), where $h \in R_{q(m,n),n}$, and

2. $\mathcal{A}_{\mathcal{D}}$ uses at most $p(m, n)$ time and outputs a string $h \in R_{q(m,n),n}$ such that $h \equiv c$, i.e., h and c are equivalent representations of the same Boolean function.

In the above definition of learnability of DNF formulas, the learner is allowed to hypothesize any DNF formula h such that the number of terms in h is polynomial in the number of terms in the target formula. That is, we allow a polynomial "blow up." We say that the class of m-term DNF formulas is *properly* learnable in the exact model if the polynomial q in the definition above satisfies $q(m, n) = m$, i.e., the learning algorithm is allowed to use only hypotheses with no more terms than appear in a shortest representation of the unknown concept to be learned. We are interested in the limit of proper learnability of m-term DNF formulas, i.e., the most number of terms (expressed as a function of n) that will permit proper learnability. This motivates the following definition.

The class $\mathcal{D}_{f()} = \cup_{n\geq 1} \cup_{0\leq m\leq f(n)} \mathcal{D}_{m,n}$ is *properly learnable in the exact model* if there exists an algorithm $\mathcal{A}$ and a polynomial function $p()$ such that for each $c \in \mathcal{D}_{m,n}$, where $0 \leq m \leq f(n)$:

1. $\mathcal{A}$ uses only membership queries and equivalence queries of the form EQ(h), where $h \in R_{m,n}$, and
2. $\mathcal{A}$ uses at most $p(m, n)$ time and outputs a string $h \in R_{m,n}$ such that $h \equiv c$.

Similar definitions can be worked out for the proper learnability of the classes $\mathcal{D}$ and $\mathcal{D}_{f()}$ in the extended PAC model. Note that all these definitions imply the following "inclusion" property: If $\mathcal{D}_{f()}$ is properly learnable (in either model), then for any function $g()$ that satisfies $g(n) \leq f(n)$ for all n, $\mathcal{D}_{g()}$ is also properly learnable. Contrapositively, to show that $\mathcal{D}_{f()}$ is not properly learnable, it suffices to consider a class $\mathcal{D}_{g()}$ where $g(n) \leq f(n)$ for all n, and show that $\mathcal{D}_{g()}$ is not properly learnable.

Since we are interested only in proper learning in this paper, we shall henceforth omit all references to the hypothesis class for both exact and PAC learning algorithms. Further, since all concept classes examined in this paper have natural representations, we also omit formal definitions of the representation for these classes in the paper.

3. General Techniques

The general technique for proving hardness results for proper learning in the exact model can be summarized as follows: Let $\mathcal{C}$ be a class of Boolean formulas and f a Boolean formula that is not necessarily in $\mathcal{C}$. If $\mathcal{C}$ can be properly learned in the exact model and there exists a polynomial time algorithm to decide if any given formula c in $\mathcal{C}$ is equivalent to f, then the question "Can f be represented in $\mathcal{C}$?" can be answered in polynomial time. If it can also be shown that the problem of deciding if f is representable in $\mathcal{C}$ is NP-Hard, then we will have effectively shown that P = NP. In other words, if P $\neq$ NP, there exists no algorithm for learning the class $\mathcal{C}$ exactly and properly.

This technique can be seen as strengthening the technique of Aizenstein, Hellerstein, and Pitt (1992) in a fairly obvious way: the latter technique does not require that the formula f be testable in polynomial time for equivalence with any given formula c in $\mathcal{C}$. Instead, a nondeterministic guess is used to find an assignment x such that $f(x) \neq c(x)$. Consequently the hardness result achievable using the latter technique needs the stronger assumption that NP $\neq$ co-NP. Interestingly, there is a more subtle reason for preferring the former "P $\neq$ NP" technique whenever possible and it is this. As will be seen, hardness results using the P $\neq$ NP technique are "scalable" in the following sense: we can replace the assumption "P $\neq$ NP" with a stronger assumption like "NP is not contained in DTIME($n^{\log n}$)" to get a stronger negative result. In contrast, the assumption "NP $\neq$ co-NP" does not scale so nicely.

We now formalize the "P $\neq$ NP" technique.

Definition 1 Let $\mathcal{C} = \bigcup_{n \geq 1} \mathcal{C}_n$ and $\mathcal{F} = \bigcup_{n \geq 1} \mathcal{F}_n$ be classes of Boolean formulas such that for each $n \geq 1$, $\mathcal{C}_n$ and $\mathcal{F}_n$ are sets of formulas over a set V_n of n Boolean variables. The class $\mathcal{F}$ is *testable with respect to* $\mathcal{C}$ if there exists an algorithm $\mathcal{A}_{\mathcal{F}}$ and a polynomial function $t()$, such that if $\mathcal{A}_{\mathcal{F}}$ is given formulas $c \in \mathcal{C}_n$ and $f \in \mathcal{F}_n$ as input, $\mathcal{A}_{\mathcal{F}}$ halts in at most $t(n, |c|, |f|)$ units of time and outputs one of the following:

1. "No," and an assignment x to the n Boolean variables such that $f(x) \neq c(x)$.
2. "Yes," and a formula $c' \in \mathcal{C}_n$ such that $f \equiv c'$ (i.e., f and c' represent the same Boolean function).

Note carefully that this definition of testability is a little weaker than the requirement mentioned in the introductory remarks to this section; in particular, item 2 of the definition does not require the formula c' to be logically equivalent to the formula c input to $\mathcal{A}_{\mathcal{F}}$: it is possible that $c \not\equiv f$ but $\mathcal{A}_{\mathcal{F}}$ outputs "Yes" and $c' \in \mathcal{C}_n$ such that $c' \equiv f$. In other words, the ability to test any $f \in \mathcal{F}_n$ for equivalence against any $c \in \mathcal{C}_n$, though sufficient to prove testability, is not quite necessary. This hair-splitting makes a difference only in Section 6, where the hardness proof for read-thrice DNF formulas is made technically easier because of the weakening of the definition of testability.

We use the following problem of deciding whether a formula can be represented in $\mathcal{C}$.

REP($\mathcal{C}$): **Representability of formulas in $\mathcal{C}$**

Instance: A set $V = \{v_1, v_2, \ldots, v_n\}$ of n Boolean variables and a formula f over V.

Question: Is there a formula $f' \in \mathcal{C}_n$, such that $f' \equiv f$?

We say that *REP($\mathcal{C}$) $\in$ P for a class $\mathcal{F}$ of Boolean formulas* if there exists an algorithm $\mathcal{A}$ such that if $\mathcal{A}$ is given any formula $f \in \mathcal{F}_n$ as input, $\mathcal{A}$ decides if there exists a $c \in \mathcal{C}_n$ such that $f \equiv c$ in time at most polynomial in $|f|$ and n. We say *REP($\mathcal{C}$) is NP-Hard for class $\mathcal{F}$* if for each problem Π in NP, Π is polynomial-time reducible to whether $f \in \mathcal{F}$ is representable in $\mathcal{C}$.

As in (Aizenstein, Hellerstein & Pitt, 1992), the technical requirement of polynomial-time recognizability of $\mathcal{C}$ (but not $\mathcal{F}$) is a necessary pre-condition for the following theorem. A class $\mathcal{C} = \bigcup_{n \geq 1} \mathcal{C}_n$ of Boolean formulas is *polynomial-time recognizable* if there exists a polynomial time algorithm such that, given a formula h and an integer n, the algorithm decides if h is an element of $\mathcal{C}_n$. The classes for which we prove hardness results have simple polynomial time algorithms for recognition. Hence, this requirement will not be explicitly mentioned in our later applications of this general technique.

THEOREM 1 *Let $C = \bigcup_{n\geq 1} C_n$ be a polynomial-time recognizable class of Boolean formulas and p() a polynomial function such that for each $c \in C_n$, $|c| \leq p(n)$. Let $\mathcal{F} = \bigcup_{n\geq 1} \mathcal{F}_n$ be a class of Boolean formulas that is testable with respect to C. If C is properly learnable in the exact model, then REP(C)∈P for the class $\mathcal{F}$.*

Proof: Suppose there exists an algorithm $\mathcal{A}_C$ that properly learns C in the exact model. Let $q()$ be a polynomial function such that $\mathcal{A}_C$ uses no more that $q(n, p(n))$ time for any target concept $c \in C_n$. We create a new algorithm $\mathcal{A}_\mathcal{R}$ such that if $\mathcal{A}_\mathcal{R}$ is given any formula $f \in \mathcal{F}_n$ as input, $\mathcal{A}_\mathcal{R}$ decides if there exists a $c \in C_n$ such that $f \equiv c$ in time at most polynomial in $|f|$ and n. The algorithm $\mathcal{A}_\mathcal{R}$ runs $\mathcal{A}_C$ to simulate the learning of f as follows.

If $\mathcal{A}_C$ makes a membership query with assignment β, $\mathcal{A}_\mathcal{R}$ returns the evaluation $f(\beta)$.

If $\mathcal{A}_C$ makes an equivalence query with a hypothesis h, $\mathcal{A}_\mathcal{R}$ checks if $h \in C_n$. Since C is polynomial time recognizable, this can be done in time polynomial in $|h|$ and n. If $h \notin C_n$ then $\mathcal{A}_\mathcal{R}$ outputs that there is no $f' \in C_n$ such that $f \equiv f'$ and halts. If h is in C_n, then $\mathcal{A}_\mathcal{R}$ runs the testing algorithm $\mathcal{A}_\mathcal{F}$ with inputs h and f. Again, this can be done in time polynomial in $|h|$, $|f|$, and n since $\mathcal{F}$ is testable with respect to C. If $\mathcal{A}_\mathcal{F}$ outputs "Yes" and a certificate $h' \in C_n$ such that $f \equiv h'$, then $\mathcal{A}_\mathcal{R}$ halts after outputting that f can be represented in C_n. On the other hand, if $\mathcal{A}_\mathcal{F}$ outputs "No" and an assignment β such that $h(\beta) \neq f(\beta)$, then clearly $f \not\equiv h$. $\mathcal{A}_\mathcal{R}$ returns the assignment β as a counterexample to $\mathcal{A}_C$.

Finally, if algorithm $\mathcal{A}_C$ exceeds $q(n, p(n))$ units of time, $\mathcal{A}_\mathcal{R}$ halts and outputs that f is not representable in the class C_n.

To prove the correctness of $\mathcal{A}_\mathcal{R}$, first suppose that f is representable in the class C_n. Now $\mathcal{A}_C$ must always ask equivalence queries with hypotheses from C_n, and within $q(n, p(n))$ steps output a representation c of f in C_n. At least when this happens (and perhaps even sooner) the testing algorithm $\mathcal{A}_\mathcal{F}$ has no choice but to output "Yes" and (possibly a different) representation c' of f in C_n. So $\mathcal{A}_R$ will make the correct decision if f is indeed representable in C_n.

Next, suppose that f is not representable in C_n. Now one of the following must happen: (i) $\mathcal{A}_C$ always hypothesizes formulas in C_n—since f cannot be represented in C_n, the testing algorithm $\mathcal{A}_\mathcal{F}$ has no choice but to output "No" for each hypothesis h from $\mathcal{A}_C$ and give an assignment β such that $h(\beta) \neq f(\beta)$ (ii) $\mathcal{A}_C$ produces a hypothesis $h \notin C_n$ as input to an equivalence query. In the first case, $\mathcal{A}_\mathcal{R}$ will correctly decide that f is not representable in C_n when $\mathcal{A}_C$ eventually exceeds $q(n, p(n))$ units of time; in the second case, $\mathcal{A}_\mathcal{R}$ makes the correct decision right away.

All the steps involved, including the simulation of $\mathcal{A}_C$ and potentially a polynomial number of runs of $\mathcal{A}_\mathcal{F}$ and a polynomial number of tests for recognizing that hypotheses used by $\mathcal{A}_C$ are indeed in C_n, can be carried out in time polynomial in $|f|$ and n. Therefore, we can conclude that REP(C) ∈ P for $\mathcal{F}$. ■

COROLLARY 1 *Let $C = \bigcup_{n\geq 1} C_n$ be a class of polynomial-time recognizable Boolean formulas and p() a polynomial function such that for each $c \in C_n$, $|c| \leq p(n)$. If there exists a class $\mathcal{F} = \bigcup_{n\geq 1} \mathcal{F}_n$ that is testable with respect to C and REP(C) is NP-Hard for class $\mathcal{F}$, then C cannot be properly learned in the exact model unless P = NP.*

Proof: If $\mathcal{C}$ is properly learnable in the exact model, then by Theorem 1, REP($\mathcal{C}$)$\in$P for $\mathcal{F}$. If in addition REP($\mathcal{C}$) is NP-Hard for $\mathcal{F}$, then by the definition of NP-Hardness, all problems in NP can be solved in polynomial time. □

Our technique for proving hardness results in the PAC model is similar to the ones used by Pitt and Valiant (1988) and Kearns, Li, Pitt, and Valiant (1987) that are built on the assumption that RP $\neq$ NP. The difference is that these other techniques apply to proving hardness results in the PAC model, whereas we want to prove hardness in the extended PAC model where membership queries are allowed. We solve the problem by combining the P $\neq$ NP technique of Theorem 1 with the ideas behind these RP $\neq$ NP techniques.

The general idea for proving negative results in the extended PAC model can be summarized as follows: Let $\mathcal{C}$ be a class of Boolean formulas and f a Boolean formula not necessarily in $\mathcal{C}$. If (i) $\mathcal{C}$ can be learned in the extended PAC model, and (ii) there exists some set X_f of examples of f which can be computed in polynomial time such that the existence of any concept in $\mathcal{C}$ that agrees with f over X_f is sufficient to guarantee that there exists a formula in $\mathcal{C}$ equivalent to f, then the decision problem of whether f can be represented in $\mathcal{C}$ is in RP. In addition, if it can be shown that this decision problem is NP-Hard, then we will have effectively shown that RP = NP.

In applying the technique to the extended PAC model, our earlier requirement of testability of f with respect to $\mathcal{C}$ has essentially been replaced with a requirement of f's "compressibility" into a set X_f with respect to $\mathcal{C}$. Formally,

Definition 2 Let $\mathcal{C} = \bigcup_{n\geq 1} \mathcal{C}_n$ and $\mathcal{F} = \bigcup_{n\geq 1} \mathcal{F}_n$ be classes of Boolean formulas such that for each $n \geq 1$, $\mathcal{C}_n$ and $\mathcal{F}_n$ are sets of Boolean formulas over a set V_n of n Boolean variables. The class $\mathcal{F}$ is *compressible with respect to* $\mathcal{C}$ if there exists an algorithm $\mathcal{A}_{\mathcal{F}}$ and a polynomial function $p()$ such that given input $f \in \mathcal{F}_n$, $\mathcal{A}_{\mathcal{F}}$ outputs in time $p(n, |f|)$ a set X_f of assignments with the following property: If there exists a formula $g \in \mathcal{C}_n$ such that for all $x \in X_f$, $g(x) = f(x)$ then there exists a formula $g' \in \mathcal{C}_n$ such that $g' \equiv f$.

Note that the definition implies that X_f must be of size at most a polynomial in $|f|$ and n.

We say that *REP($\mathcal{C}$) $\in$ RP for a class $\mathcal{F}$ of Boolean formulas* if there exists a randomized algorithm $\mathcal{A}$ such that if $\mathcal{A}$ is given any formula $f \in \mathcal{F}_n$ then

1. If there exists $c \in \mathcal{C}_n$ such that $f \equiv c$ then $\mathcal{A}$ outputs "Yes" with probability at least $\frac{3}{4}$ and "No" with probability at most $\frac{1}{4}$, and

2. If there exists no $c \in \mathcal{C}_n$ such that $f \equiv c$ then $\mathcal{A}$ outputs "No".

Moreover, $\mathcal{A}$ halts in time at most polynomial in $|f|$ and n.

We have the following general result for proper learnability in the extended PAC model:

THEOREM 2 *Let $\mathcal{C} = \bigcup_{n\geq 1} \mathcal{C}_n$ be a polynomial-time recognizable class of Boolean formulas and $r()$ a polynomial function such that for all $c \in \mathcal{C}_n$, $|c| \leq r(n)$. Let $\mathcal{F} = \bigcup_{n\geq 1} \mathcal{F}_n$ be a class of Boolean formulas that is compressible with respect to $\mathcal{C}$. If $\mathcal{C}$ is properly learnable in the extended PAC model, then REP($\mathcal{C}$)$\in$RP for the class $\mathcal{F}$.*

Proof: Suppose there exists an algorithm $\mathcal{A_C}$ that properly learns $\mathcal{C}$ in the extended PAC model. Let $s()$ be a polynomial function such that for any target concept $c \in \mathcal{C}_n$, $\mathcal{A_C}$ uses no more that $s(n, r(n), 1/\epsilon, 1/\delta)$ time, for error parameter ϵ and confidence parameter δ. We now create a randomized algorithm $\mathcal{A_R}$ such that if $\mathcal{A_R}$ is given any formula $f \in \mathcal{F}_n$, then, in time polynomial in $|f|$ and n, $\mathcal{A}$ outputs (i) "No", if there exists no $c \in \mathcal{C}_n$ such that $f \equiv c$ (ii) "Yes" with probability at least $\frac{3}{4}$, if there exists $c \in \mathcal{C}_n$ such that $f \equiv c$. The algorithm $\mathcal{A_R}$ runs $\mathcal{A_C}$ to simulate the learning of f as follows.

Using the fact that $\mathcal{F}$ is compressible with respect to $\mathcal{C}$, $\mathcal{A_R}$ first computes a set X_f of assignments in time polynomial in $|f|$ and n. Next, a distribution D is defined over the set of all assignments as follows: for each element $x \in X_f$, $D(x) = \frac{1}{|X_f|}$ and for all other assignments y, $D(y) = 0$. That is, the distribution is uniform over X_f and 0 (zero) elsewhere. Finally, the parameters $\epsilon = \frac{1}{|X_f|+1}$ and $\delta = \frac{1}{4}$ are given as input to $\mathcal{A_C}$.

If $\mathcal{A_C}$ makes a membership query with an assignment β, $\mathcal{A_R}$ returns the evaluation $f(\beta)$. If $\mathcal{A_C}$ makes a sample query, $\mathcal{A_R}$ returns an assignment β randomly chosen according to distribution D along with its evaluation $f(\beta)$. If at any point $\mathcal{A_C}$ exceeds $s(n, r(n), 1/\epsilon, 1/\delta)$ units of time, $\mathcal{A_R}$ terminates $\mathcal{A_C}$, outputs that f is not representable in $\mathcal{C}_n$ and halts. If $\mathcal{A_C}$ terminates and outputs a hypothesis h, $\mathcal{A_R}$ first checks if $h \in \mathcal{C}_n$. Since $\mathcal{C}$ is polynomial time recognizable, this check can be done in polynomial time. If $h \notin \mathcal{C}_n$, $\mathcal{A_R}$ outputs that f is not representable in $\mathcal{C}_n$ and halts. If $h \in \mathcal{C}_n$, then $\mathcal{A_R}$ checks if $h(x) = f(x)$ for all assignments $x \in X_f$. Again, this check can be done in polynomial time since $|X_f|$ is at most a polynomial in $|f|$ and n. If h and f agree on the evaluation of all the assignments in X_f, the representation question of f is answered in the affirmative, else $\mathcal{A_R}$ outputs that f is not representable in $\mathcal{C}_n$.

If f is representable in the class $\mathcal{C}_n$, then there exists at least one formula in $\mathcal{C}_n$ that agrees with f on the evaluation of all the assignments in X_f. Since $\epsilon = \frac{1}{|X_f|+1}$, $\delta = \frac{1}{4}$, and D is uniform over X_f and 0 elsewhere, with probability at least $1 - \delta = \frac{3}{4}$, $\mathcal{A_C}$ outputs a hypothesis g that has error rate no greater than ϵ, i.e., g agrees with f on all the assignments in X_f. Even if g is not equivalent to f, the compressibility of $\mathcal{F}$ with respect to $\mathcal{C}$ is enough evidence for $\mathcal{A_R}$ to correctly conclude that f is representable in $\mathcal{C}_n$. In other words, if f is representable in $\mathcal{C}_n$, then $\mathcal{A_R}$ will correctly answer "Yes" with probability at least $\frac{3}{4}$.

If f is not representable in $\mathcal{C}$, then no such formula g exists. In such a case, $\mathcal{A_C}$ does one of the following: (i) it uses more than $s(n, r(n), 1/\epsilon, 1/\delta)$ time, (ii) it outputs a hypothesis $h \notin \mathcal{C}_n$, or (iii) it outputs a hypothesis h that does not agree with f on the evaluation of all assignments in X_f. In all three cases the failure of algorithm $\mathcal{A_C}$ will be detected by $\mathcal{A_R}$. Therefore, $\mathcal{A_R}$ always correctly answers "No" if f is not representable in $\mathcal{C}_n$.

Finally, the total time taken by algorithm $\mathcal{A_R}$ (including the simulation of $\mathcal{A_C}$, the computation of X_f, the test to check if the hypothesis $h \in \mathcal{C}_n$ and testing whether a hypothesis of $\mathcal{A_C}$ agrees with f on all the examples in X_f) is at most a polynomial in $|f|$ and n. Therefore, $\mathcal{A_R}$ is a randomized polynomial time algorithm for REP($\mathcal{C}$) for formulas in $\mathcal{F}$.

■

Note that the algorithm $\mathcal{A_R}$ is not necessarily able to produce a formula $f' \in \mathcal{C}_n$ equivalent to f even if it decides that f can indeed be represented in $\mathcal{C}_n$. In contrast, the proof of Theorem 1 in the exact model yields an algorithm that can produce such a representation.

COROLLARY 2 *Let $C = \bigcup_{n\geq 1} C_n$ be a class of polynomial-time recognizable Boolean formulas and $r()$ a polynomial function such that for each $c \in C_n$, $|c| \leq r(n)$. If there exists a class $\mathcal{F} = \bigcup_{n\geq 1} \mathcal{F}_n$ of Boolean formulas that is compressible with respect to C and REP(C) is NP-Hard for $\mathcal{F}$, then C is not properly learnable in the extended PAC model unless RP = NP.*

Proof: If C is properly learnable in the extended PAC model, then by Theorem 2, REP(C)$\in$RP for the class $\mathcal{F}$. If in addition REP(C) is NP-Hard for $\mathcal{F}$, then by the definition of NP-Hardness, all problems in NP can be solved in randomized polynomial time. Thus, if C is properly PAC learnable with membership queries then RP=NP. □

4. Proper Exact Learnability of m-term DNF Formulas

Let $\mathcal{D}_{m,n}$ denote the class of DNF formulas over n variables that have at most m terms. We use $\mathcal{D}_{\phi()}$ to denote the class $\bigcup_{n\geq 1} \mathcal{D}_{\phi(n),n}$ of DNF formulas. Using the general technique of the previous section we first prove that under the assumption that P $\neq$ NP, the class $\mathcal{D}_{\phi()}$, where $\phi(n) = \frac{13n}{6}$, is not properly learnable in the exact model. Next, we improve on this by proving a general result which has the following flavor: m-term DNF formulas over n variables are hard to learn exactly and properly, assuming NP is not contained in DTIME($f(m,n)$). Here $f(m,n)$ is a functional inverse of m with respect to n—its precise definition will become clear later. This general result shows the "scalable" aspect of our technique for exact and proper learning. By using stronger assumptions than P $\neq$ NP, we can get stronger negative results.

All our reductions in this and the next 2 sections are from the following problem, which is a variant of the exact cover problem. Actually, Garey and Johnson (Garey & Johnson, 1979) cite a slightly different version as being NP-complete. (The requirement below that each element of X occur in "exactly 3" sets in C is replaced with "at most 3" in Garey and Johnson.) However, it is not too difficult to prove that our variant is also NP-complete (see (Pillaipakkamnatt & Raghavan, 1994) for a proof).

Problem: **Exact Cover by 3-Sets Where Each Element Occurs 3 Times (X3C3)**

Instance: A set X such that $|X| = 3q$ and a collection C of 3-element subsets of X such that each element of X occurs in exactly 3 sets of C.

Question: Does there exist $C' \subseteq C$ such that each element of X appears in exactly one set of C'?

THEOREM 3 *X3C3 is NP-Complete.*

At this stage, it is convenient to introduce the following notation, used in the rest of the paper. Let V be a set of Boolean variables. For any assignment $\beta : V \rightarrow \{False, True\}$, $b \in \{False, True\}$ and $x \in V$, the assignment $\beta_{x\leftarrow b}$ is the assignment obtained by setting the value of the variable x to b and setting all other variables to the values in β. Let **False** be the assignment that assigns *False* to all variables in V, i.e., $\forall v \in V$, $\mathbf{False}(v) = False$. We view a term t of a DNF formula as a set of literals since this facilitates addition and deletion of literals to obtain new terms.

THEOREM 4 *There exists a class $\mathcal{F}$ of Boolean formulas testable with respect to $\mathcal{D}_{\phi()}$, where $\phi(n) = \frac{13n}{6}$, such that REP$(\mathcal{D}_{\phi()})$ is NP-Hard for $\mathcal{F}$.*

Proof: We use a reduction from X3C3. The class $\mathcal{F} = \{g_{\langle X,C\rangle} : \langle X, C\rangle$ is an instance of X3C3 $\}$ contains only DNF formulas and is defined as follows. Let $\langle X, C\rangle$ be an instance of X3C3, where $X = \{x_1, x_2, \ldots, x_{3q}\}$ and $C = \{c_1, c_2, \ldots, c_{3q}\}$. The formula $g = g_{\langle X,C\rangle}$ is defined over the set of variables $V = X \cup C$. When viewed as a function, g evaluates an assignment β to *True* if and only if β is one of the assignments in the following list:

1. **False**

2. For each $x_i \in X$, $\mathbf{False}_{x_i \leftarrow \mathit{True}}$

3. For each $c_i \in C$, the four assignments where at least two of the three x's in c_i are assigned *True*. That is, if $c_i = \{x_j, x_k, x_l\}$ the assignments:
   (i) $\mathbf{False}_{x_j \leftarrow \mathit{True}, x_k \leftarrow \mathit{True}, x_l \leftarrow \mathit{True}}$
   (ii) $\mathbf{False}_{x_j \leftarrow \mathit{True}, x_k \leftarrow \mathit{True}}$
   (iii) $\mathbf{False}_{x_k \leftarrow \mathit{True}, x_l \leftarrow \mathit{True}}$
   (iv) $\mathbf{False}_{x_j \leftarrow \mathit{True}, x_l \leftarrow \mathit{True}}$

4. For each $c_i \in C$, $\mathbf{False}_{c_i \leftarrow \mathit{True}}$

5. For each $c_i \in C$, the four assignments where c_i and at least two of the three x's in c_i are assigned *True*. That is, if $c_i = \{x_j, x_k, x_l\}$ the assignments:
   (i) $\mathbf{False}_{c_i \leftarrow \mathit{True}, x_j \leftarrow \mathit{True}, x_k \leftarrow \mathit{True}, x_l \leftarrow \mathit{True}}$
   (ii) $\mathbf{False}_{c_i \leftarrow \mathit{True}, x_j \leftarrow \mathit{True}, x_k \leftarrow \mathit{True}}$
   (iii) $\mathbf{False}_{c_i \leftarrow \mathit{True}, x_k \leftarrow \mathit{True}, x_l \leftarrow \mathit{True}}$
   (iv) $\mathbf{False}_{c_i \leftarrow \mathit{True}, x_j \leftarrow \mathit{True}, x_l \leftarrow \mathit{True}}$

The formula g evaluates no more than $30q + 1$ assignments to *True*. (The assignment **False** is one, assignments in items 3 and 5 add at most $24q$ more since there are $3q$ sets in C, assignments in item 2 add $3q$ more since there are $3q$ elements in X, and assignments in item 4 add another $3q$ more.) Therefore this is a polynomial transformation. Fix the DNF formula representation of $g = g_{\langle X,C\rangle}$ by simply making each of the above list of at most $30q + 1$ assignments a term in the natural way. To complete the proof of the theorem, it suffices to show that:

1. $\mathcal{F}$ is testable with respect to $\mathcal{D}_{\phi()}$, where $\phi(n) = \frac{13n}{6}$, and

2. If $g = g_{\langle X,C\rangle}$ is any formula in $\mathcal{F}_{6q}$, then g can be represented as a $13q$-term DNF formula if and only if $\langle X, C\rangle$ has an exact cover.

To prove (1), we develop an algorithm, $\mathcal{A}$, such that if $\mathcal{A}$ is given formulas $h \in \mathcal{D}_{13q,6q}$ and $g = g_{\langle X,C\rangle} \in \mathcal{F}_{6q}$ as input, $\mathcal{A}$ halts in time polynomial in $|h|$, $|g|$ and q and outputs (i) "No" and an assignment x such that $h(x) \neq g(x)$ or (ii) "Yes", if $h \equiv g$. Since both h and g are of size polynomial in q, it suffices to show that $\mathcal{A}$ needs only a polynomial in q units of time.

Now, $g \equiv h$ if and only if both $g \Rightarrow h$ and $h \Rightarrow g$ hold. Let P be the set of at most $30q + 1$ assignments that g evaluates to *True*. To test if $g \Rightarrow h$ holds, $\mathcal{A}$ simply checks if all of the assignments in P are evaluated to *True* by h as well. Clearly, this can be done in time polynomial in q and a counterexample, if one exists, can be found. Next, observe that $h \Rightarrow g$ if and only if for every term t in h, the set $Q(t)$ of assignments evaluated to *True* by t satisfies $Q(t) \subseteq P$. Therefore, $\mathcal{A}$ can test if $h \Rightarrow g$ by considering each term t of h, generating the assignments in $Q(t)$ and checking if every such assignment generated is in P. Algorithm $\mathcal{A}$ stops the generation of $Q(t)$ for any term t when either all of the assignments in $Q(t)$ have been generated or $\mathcal{A}$ finds an assignment in $Q(t)$ that is not in P. In the latter case, such an assignment is a certificate that $h \not\Rightarrow g$ and therefore that $h \not\equiv g$. Since h has at most $13q$ terms and $\mathcal{A}$ generates at most $|P| + 1 \leq 30q + 2$ assignments per term, $\mathcal{A}$ runs in time polynomial in q.

We prove (2) through a series of claims. Let $g = g_{\langle X,C\rangle}$ be a formula in $\mathcal{F}_{6q}$ and let $V = X \cup C$. Let $\mathbf{Z}$ denote the term $\{\overline{v} : v \in V\}$. We define the sets S_1 and S_2 of terms over literals that correspond to variables in V as follows:

$$\begin{aligned} S_1 &= \{\mathbf{Z} - \{\overline{x}_j, \overline{x}_k, \overline{x}_l\} : c_i = \{x_j, x_k, x_l\} \text{ is a set in } C\}. \\ S_2 &= \{\mathbf{Z} - \{\overline{c}_i\},\quad (\mathbf{Z} \cup \{x_j, x_k\}) - \{\overline{c}_i, \overline{x}_j, \overline{x}_k, \overline{x}_l\}, \\ &\qquad (\mathbf{Z} \cup \{x_k, x_l\}) - \{\overline{c}_i, \overline{x}_j, \overline{x}_k, \overline{x}_l\},\quad (\mathbf{Z} \cup \{x_l, x_j\}) - \{\overline{c}_i, \overline{x}_j, \overline{x}_k, \overline{x}_l\} : \\ &\qquad\quad c_i = \{x_j, x_k, x_l\} \text{ is a set in } C\}. \end{aligned}$$

CLAIM 1 *The set $S = S_1 \cup S_2$ contains exactly the prime implicants of g.*

Proof: It can be verified that each term in S is indeed an implicant of g. We prove that each term in S is a prime implicant. Assume to the contrary that for some term t in S, there exists $t' \subset t$ such that t' is an implicant of g. It is sufficient to consider the case where $t' = t - \{x\}$ for some literal x in t.

Case 1: $t \in S_1$.
No positive assignment of g (i.e., none of the at most $30q + 1$ assignments) satisfies more than three variables from X. Since t is in S_1, it contains one literal per variable in V, except for three missing literals that correspond to variables in X. Thus, x cannot be a literal that corresponds to a variable in X, lest t' be satisfied by an assignment that satisfies more than three variables from X. But if x corresponds to a variable in C, then t' would be satisfied by an assignment that satisfies exactly one variable in C and one variable in X. No such assignment exists in our set of positive assignments. Hence, each term in S_1 is a prime implicant.

Case 2: $t \in S_2$.
Deleting any literal in t that corresponds to a variable in C produces a term that is satisfied by an assignment that satisfies two variables from C. No such assignment exists in our function g. Hence, it is sufficient to consider the case where literal x corresponds to a variable in X. If t is a term of the form $\mathbf{Z} - \{\overline{c}_i\}$, and x corresponds to a variable in X, then t' is satisfied by an assignment that satisfies one variable in C and one variable in X. Hence t' is not an implicant of g. For a term t of the form $(\mathbf{Z} \cup \{x_j, x_k\}) - \{\overline{c}_i, \overline{x}_j, \overline{x}_k, \overline{x}_l\}$, any assignment that satisfies t satisfies at least two variables and at most three variables in

X. If x is a negated literal, then t' is satisfied by an assignment that satisfies more than three variables in X. If x is a positive literal, then t' is satisfied by an assignment that satisfies exactly one variable in X and one variable in C. Thus, every term in S_2 is a prime implicant.

Finally, we need to show that g has no other prime implicants. The crucial observation here is this. Any implicant t of g is satisfied by at least one assignment in P. Now t must contain at least $6q - 4$ complemented literals, lest it be satisfied by some assignment not in P. (All assignments in P set at least $6q - 4$ variables to *False*.) Similarly, there must be some $c_i = \{x_j, x_k, x_l\}$ in C such that t has all the negated literals over the variables in $Y = V - \{c_i, x_j, x_k, x_l\}$, lest it satisfy some assignment not in P. Now, the projection of g obtained by setting all the variables in Y to false is precisely the formula $\overline{c}_i + x_jx_k + x_kx_l + x_lx_j + \overline{x}_j\overline{x}_k\overline{x}_l$, all five terms of which are prime implicants of the projection. Therefore, t must be a superset of one of the five corresponding terms in $S_1 \cup S_2$.

□

Since $|S| = 15q$, g can be actually be represented using $15q$ or fewer terms.

CLAIM 2 *If $\langle X, C\rangle$ has an exact cover, then g can be represented using $13q$ terms.*

Proof: Let $C' \subset C$ be an exact cover of $\langle X, C\rangle$. Let $S_1' = \{\mathbf{Z} - \{\overline{x}_j, \overline{x}_k, \overline{x}_l\} : \{x_j, x_k, x_l\}$ is a set in $C'\}$. Now $|S_1'| = |C'| = q$, since X contains $3q$ elements and each set in C' contains exactly 3 elements. Consider the DNF formula $f = \bigvee_{t \in S_1'} t \vee \bigvee_{t \in S_2} t$. By Claim 1, f evaluates a subset of the positive assignments of g to *True*. In particular, all assignments in items 1,3,4 and 5 of the definition of g are evaluated correctly using the terms in S_2. Since C' is an exact cover of X, for each $x \in X$, there exists $c \in C'$ such that $x \in c$. Thus, for each assignment in item 2 of the definition of g, there exists a term in S_1' that evaluates β_x to *True*. The number of terms in f is $|S_1'| + |S_2| = 13q$.

□

CLAIM 3 *If g can be represented as a DNF formula f with at most $13q$ terms, then there exists an exact cover for $\langle X, C\rangle$.*

Proof: Suppose there exists a DNF formula f with at most $13q$ terms such that $f \equiv g$. Without loss of generality, assume that f contains a set of prime implicants from S. (Since every term t of f is an implicant of g and therefore a superset of one of the prime implicants in S, we can replace each term t in f by a term $t' \in S$ such that $t' \subseteq t$. The resultant formula will still be equivalent to g.)

Now, all terms in S_2 must be in f, lest some assignment in part 4 or 5 of the definition of g be not evaluated to *True* by f. These account for $12q$ terms. The remaining at most q terms must be a set $S_1' \subset S_1$. Every prime implicant in S_1 is satisfied by exactly 3 assignments in item 2 of the list of assignments satisfied by g. Moreover, only the prime implicants in S_1 can evaluate these assignments to *True*. Since there are $3q$ such assignments, it follows that there must be at least q terms in S_1' and $C' = \{\{x, y, z\} : \mathbf{Z} - \{\overline{x}_j, \overline{x}_k, \overline{x}_l\} \in S_1'\}$ contains q sets of pairwise disjoint sets. By the definition of S_1, $C' \subseteq C$ and therefore C' is an exact cover.

□

From the proof of the claim above, g requires at least $13q$ terms to be represented as a DNF formula, and if $\langle X, C\rangle$ has an exact cover, then g can be represented with exactly $13q$

terms. Since the number of variables over which g is defined is $n = 6q$, the number of terms in g is $13q = \frac{13n}{6}$ (if $\langle X, C\rangle$ has an exact cover).

Since $\mathcal{F}$ is testable with respect to $\mathcal{D}_{\phi()}$, $\phi(n) = \frac{13n}{6}$, and, by Claims 2 and 3, there exists an exact cover for $\langle X, C\rangle$ if and only if $g = g_{\langle X,C\rangle} \in \mathcal{F}_{6q}$ can be represented as a DNF formula with exactly $13q$ terms, the theorem follows. ■

COROLLARY 3 *If P ≠ NP, then* $\frac{13n}{6}$*-term DNF formulas over* n *variables are not properly learnable in the exact model.*

Proof: Immediate from Corollary 1 and Theorem 4. □

By the definition of proper learnability, $m(n)$-term DNF formulas, for any $m(n) > \frac{13n}{6}$, are also not properly and exactly learnable unless P = NP. We now give a general hardness result for $m(n)$-term DNF formulas, which is particularly useful for functions $m(n)$ that are $o(n)$. In what follows, we adopt the following definition of a functional inverse: $f^{-1}(n)$ is the least non-negative integer j such that $f(j) \geq n$. (If no such integer exists then $f^{-1}(n)$ is infinity.) Following standard practice, we use the shorthand notation $poly(f(n))$ to denote functions that are at most a polynomial in $f(n)$.

A technical requirement needed to prove the following theorem is that the real function $f(n)$ be efficiently computable in the following weak sense. A total function $f : \mathcal{R} \to \mathcal{R}$ is *computable in pseudo-polynomial time* if there exists an algorithm $\mathcal{A}_f$ that, when given integers n and N as input, can decide if $f(N) \geq n$ in time polynomial in n and N.

THEOREM 5 *If* $f()$ *is computable in pseudo-polynomial time and* $\mathcal{D}_{f()}$ *is properly learnable in the exact model, then NP* $\subseteq$ *DTIME* $(poly(max(n, f^{-1}(\frac{13n}{6}))))$.

Proof: Let $\mathcal{F} = \{g'_{\langle X,C\rangle} : \langle X, C\rangle \text{ is an instance of X3C3}\}$ be a class of DNF formulas, where for each instance $\langle X, C\rangle$ of X3C3, the formula $g'_{\langle X,C\rangle}$ is as defined below.

Let $\langle X, C\rangle$ be any arbitrary instance of X3C3, with $X = \{x_1, x_2, \ldots, x_{3q}\}$ and $C = \{c_1, c_2, \ldots, c_{3q}\}$. Let g be the Boolean formula defined in Theorem 4 over the set of variables $V = X \cup C$. Let $f()$ be a real function that is computable in pseudo-polynomial time. In what follows, assume that the range of $f()$ is not bounded from above by any fixed integer c, otherwise $f^{-1}(n) = \infty$, for $n > c$, and the theorem is trivially true for such a function $f()$. Compute $f^{-1}(13q)$ as follows. By iterating through integer values $j \geq 0$, determine the least j such that $f(j) \geq 13q$.

Let $N = \max(6q, f^{-1}(13q))$. If $N > 6q$, then create a set of $N - 6q$ new variables $Y = \{y_1, y_2, \ldots, y_{N-6q}\}$; if $N = 6q$, let $Y = \emptyset$. We define $g'_{\langle X,C\rangle}$ over the set of variables $V' = X \cup C \cup Y$ as follows.

For any assignment β, $g'_{\langle X,C\rangle}(\beta) = \mathit{True}$ if and only if $g(\beta) = \mathit{True}$ and for all $y \in Y$, $\beta(y) = \mathit{False}$.

Given any instance $\langle X, C\rangle$ of X3C3 with $|X| = |C| = 3q$, we can compute a DNF formula $g'_{\langle X,C\rangle}$ in time polynomial in $N = \max(6q, f^{-1}(13q))$. As in the proof of Theorem 4, fix the DNF formula $g'_{\langle X,C\rangle}$ by making terms out of the at most $30q + 1$ assignments. (However, now the number of literals in $g'_{\langle X,C\rangle}$ may be more, since N is potentially larger than $6q$.)

We claim that:

1. $\mathcal{F}$ is testable with respect to $\bigcup_{q\geq 1} \mathcal{D}_{13q,\max(6q,f^{-1}(13q))}$.

2. If $g' = g'_{\langle X,C\rangle}$ is any formula in $\mathcal{F}_{\max(6q,f^{-1}(13q))}$ then g' can be represented as a $13q$-term DNF formula if and only if $\langle X, C\rangle$ has an exact cover.

To prove (1), it suffices to give an algorithm, $\mathcal{A}$, such that if $\mathcal{A}$ is given formulas $h \in \mathcal{D}_{13q,\max(6q,f^{-1}(13q))}$ and $g' = g'_{\langle X,C\rangle} \in \mathcal{F}_{\max(6q,f^{-1}(13q))}$ as input, $\mathcal{A}$ halts in time polynomial in $N = \max(6q, f^{-1}(13q))$ and outputs (i) "No" and an assignment x such that $h(x) \neq g'(x)$ or (ii) "Yes", if $h \equiv g'$.

If $N = 6q$, then $\mathcal{A}$ proceeds as in the proof of Theorem 4. So, suppose that $N = f^{-1}(13q) > 6q$ and h is defined over the set of variables $X \cup C \cup Y$, where $Y \neq \emptyset$. To check if $g' \Rightarrow h$, test (as in Theorem 4) if the set P of at most $30q + 1$ assignments that g' evaluates to *True* are evaluated to *True* by h as well. If not, one of the assignments in P is a counterexample. To check if $h \Rightarrow g'$, check if all the terms of h contain negated literals corresponding to all the variables in Y. If any particular term t does not contain a literal $\overline{y}$, where $y \in Y$, then any assignment formed by setting y to *True* and satisfying the remaining literals in t is a counterexample that satisfies h but not g'. Finally, if all the terms in h contain negated literals corresponding to all the variables in Y, then consider the DNF formula h' obtained from h by deleting all the literals corresponding to variables in Y. At this point, we can assert that $h \Rightarrow g'$ if and only if $h' \Rightarrow g$, where g is the original formula of Theorem 4 obtained from g' by deleting all literals corresponding to variables in Y. By the proof of Theorem 4 testing if $h' \Rightarrow g$ can be done in time polynomial in q. Clearly, all of the above tests can be done in time polynomial in N since we are assuming that $N > 6q$ and h and g' have only O(q) terms over N variables.

To prove (2), note that, by Theorem 4, it suffices to prove that g' can be represented as a $13q$-term DNF formula if and only if g can be represented as a $13q$-term DNF formula. The "if" part follows by noting that any DNF formula for g can be transformed to a DNF formula for g' by simply appending a set of negated literals corresponding to variables in Y to every term of g. The "only if" part follows by noting that any implicant of g' must contain all the negated literals corresponding to variables in Y. Therefore, stripping off the negated literals corresponding to variables in Y from every term of a DNF representation of g' gives a DNF representation of g.

By Theorem 1 and (1) above, the existence of a polynomial time exact and proper learning algorithm for $\mathcal{D}_{13q,N}$ for every $q > 0$ implies that recognizing if g' can be represented in $\mathcal{D}_{13q,N}$ is in DTIME($poly(N)$). Since the transformation to g' can be done in time polynomial in N, X3C3 is NP-hard, and (2) above holds, we can conclude that if there exists a polynomial time exact and proper learning algorithm for $\mathcal{D}_{13q,N}$, then all problems in NP can be solved in DTIME($poly(N)$). The theorem now follows by putting $n = 6q$. ∎

Of interest are the following corollaries, which illustrate how Theorem 5 can be used to get stronger results with stronger assumptions.

COROLLARY 4 *If P $\neq$ NP, then n^α-term DNF formulas over n variables cannot be properly learned in the exact model for any fixed constant $\alpha > 0$.*

Proof: It suffices to consider $\alpha \leq 1$ since the result follows from Theorem 4 for $\alpha > 1$. For any constant α, $0 < \alpha \leq 1$, let c be any integer such that $\frac{1}{c} < \alpha$. Put $f(n) = n^{\frac{1}{c}}$ in Theorem 5. Now $f(n)$ is certainly computable in pseudo-polynomial time. Since $f^{-1}(\frac{13n}{6}) = \lceil (\frac{13n}{6})^c \rceil$ is in $poly(n)$, the corollary follows. □

A reasonable assumption is that NP is not contained in DTIME($n^{O(\log n)}$). This assumption leads to the following.

COROLLARY 5 *If NP is not contained in DTIME($n^{O(\log n)}$), then $2^{\sqrt{\log n}}$-term DNF formulas over n variables cannot be properly learned in the exact model.*

Proof: Put $f(n) = 2^{\sqrt{\log n}}$ in Theorem 5. Again $f(n)$ is computable in pseudo-polynomial time and $f^{-1}(\frac{13n}{6}) = \left\lceil (\frac{13n}{6})^{\log(\frac{13n}{6})} \right\rceil$ is in $n^{O(\log n)}$. The corollary follows. □

Since NP is definitely contained in DTIME($2^{poly(n)}$) the next corollary uses the strongest assumption that is still plausible.

COROLLARY 6 *If there exists some constant $\epsilon > 0$ such that NP is not contained in DTIME($poly(2^{n^\epsilon})$) then there exists a constant $c > 0$ such that $\log^c n$-term DNF formulas over n variables cannot be properly learned in the exact model.*

Proof: If such an $\epsilon > 0$ does exist, let $c > \frac{1}{\epsilon}$ be any integer. Now $f(n) = (\log n)^c$ is computable in pseudo-polynomial time and $f^{-1}(\frac{13n}{6}) = \left\lceil 2^{(\frac{13n}{6})^{\frac{1}{c}}} \right\rceil$ is $o(poly(2^{n^\epsilon}))$. By Theorem 5, $\log^c(n)$-term DNF formulas cannot be learned in the exact model. □

The positive result of Bshouty, Goldman, Hancock, and Matar (1993) shows that the constant c of the last corollary is greater than $\frac{1}{2}$. It is open if this can be improved further. This last corollary is not so much a negative result as it is an indication that improved positive results for proper learnability of DNF formulas must necessarily imply a corresponding improvement in what is known about the smallest DTIME class in which NP is contained.

5. Proper PAC Learnability of m-term DNF Formulas

We now turn to proper PAC learnability of m-term DNF formulas. It must be mentioned that the results in this section imply the results for proper exact learnability if all the complexity-theoretical assumptions of this section hold. However, the assumptions in this section involve the relationship of randomized time (RTIME) to NP and are stronger than the corresponding assumptions about DTIME of the previous section. So we have taken the "safe" route by proving the results for PAC and exact learnability independently.

THEOREM 6 *There exists a class $\mathcal{F}$ of Boolean formulas compressible with respect to $\mathcal{D}_{\phi()}$, where $\phi(n) = \frac{13n}{6}$, such that REP($\mathcal{D}_{\phi()}$) is NP-Hard for $\mathcal{F}$.*

Proof: The reduction is from X3C3. Let the class $\mathcal{F}$ of DNF formulas is exactly the class defined in Theorem 4. It suffices to show that the class $\mathcal{F}$ is compressible with respect to $\mathcal{D}_{\phi()}$, since REP($\mathcal{C}$) is NP-hard for $\mathcal{F}$ (Theorem 4).

We now show that $\mathcal{F}$ is compressible with respect to $\mathcal{D}_{\phi()}$. We create an algorithm $\mathcal{A}$ such that if $\mathcal{A}$ is given as input a formula $g = g_{\langle X,C \rangle} \in \mathcal{F}_{6q}$ then it outputs, in time polynomial in $|g|$ and q, a set X_g with the following property: if there exists a formula $h \in \mathcal{D}_{13q,6q}$ such that for all $x \in X_g$, $g(x) = h(x)$, then there exists a formula $f \in \mathcal{D}_{13q,6q}$ such that $f \equiv g$. Since $|g|$ is itself polynomial in q, it suffices to show that $\mathcal{A}$ runs in time polynomial in q.

Let P be the set of at most $30q + 1$ assignments that g evaluates to *True*. We define the set X_g of assignments as follows.

$$X_g = \{\beta_{v \leftarrow b} : \ v \in V, \ b \in \{\mathit{True}, \mathit{False}\}, \beta \in P\}$$

The number of assignments in X_g is at most $(30q + 1)(6q + 1) = \mathrm{O}(q^2)$. Therefore X_g can be computed in polynomial time and is of polynomial size. It remains only to show that if there is a DNF formula h of at most $13q$ terms such that h and g agree with the assignments in X_g then there is a DNF formula of at most $13q$ terms that is equivalent to g.

Let h be a $13q$ term DNF formula consistent with the set X_g of examples. For each term t in h such that t satisfies none of the positive examples in X_g, delete t from h. Clearly, h is still consistent with X_g, and has $13q$ or fewer terms. If t is a term in h that has $6q - 4$ or fewer literals, then t can be deleted, since (i) none of the positive assignments in X_g has more than 4 variables set to *True* (ii) when a positive assignment contains 4 variables set to *True*, exactly one variable of the form c_i is set to *True* and (iii) none of the positive assignments satisfy exactly one variable of the from c_i and x_j. Thus, each remaining term has at least $6q - 3$ literals. Moreover, no term t contains more than three positive literals corresponding to variables in X. For each term t in h, if t is a superset of any of the prime implicants defined in the proof of Theorem 4, then replace t with a prime-implicant. Again, if h was consistent with X_g before, it is still consistent with all assignments in X_g. If each term in h is a prime-implicant of g, we are done.

If to the contrary, there exists a term t in h that is not a prime implicant, we show that h is not consistent with the set X_g of examples. At this juncture, each term in h has $6q - 3$ or more literals. If term t has two or more positive literals of the form c_i, or t does not contain literals corresponding to two or more variables of the the form c_i, then by the construction of X_g, one of the following is true: (i) it is satisfied by an assignment in X_g that is a negative example (ii) none of the assignments that satisfy t are in X_g. If the former is true, then we contradict the assumption that h is consistent with X_g. If the latter is true, then t should have been deleted at an earlier step. Thus t can contain at most one literal of the form c_i.

If t contains one literal of the form c_i then it must be a term of length exactly $6q$, lest it be satisfied by an assignment that satisfies exactly c_i and a variable of the form x_j (all such assignments are negative examples). But this would make t an implicant of g, contrary to our assumption.

If t is missing a literal corresponding to a variable of the form c_i and is not a prime implicant of g, then t must contain at least 2 positive literals of the form x_j and x_k, lest it be satisfied by a negative example in X_g. If x_j and x_k do not belong in the set c_i of C, then t **is satisfied by a negative example, contrary to our assumption that** h **is consistent with** X_g. **But this would make** t **a prime implicant of** g **(**t **would be in** S_2**).**

Finally, consider the case where all literals corresponding to variables in C are negated. Since t cannot be satisfied by any assignment that satisfies two variables x_j and x_k, where x_j and x_k do not belong together in any set c_i in C, t must be an implicant of h, which would contradict our assumption that all such implicants were replaced by prime implicants. ■

COROLLARY 7 *If RP $\neq$ NP, then $\frac{13n}{6}$-term DNF formulas over n variables are not properly learnable in polynomial time in the extended PAC model.*

Proof: Immediate from Corollary 2 and Theorem 6. □

It is possible to get stronger negative results by using assumptions stronger than RP $\neq$ NP. The proof of the following theorem is substantially similar to the proof of Theorem 5; so we omit the proof and the corresponding corollaries.

THEOREM 7 *If $\mathcal{D}_{f()}$ is properly learnable in the extended PAC model and $f()$ is computable in pseudo-polynomial time, then NP $\subseteq$ RTIME$(poly(max(n, f^{-1}(\frac{13n}{6}))))$.*

6. Read-Thrice DNF Formulas

We strengthen the result in (Aizenstein, Hellerstein & Pitt, 1992) to show that read-thrice DNF formulas are hard to learn, given the weaker assumption that P$\neq$NP. In addition we also show that, under the assumption that RP$\neq$NP, read-thrice DNF formulas are not properly learnable in the PAC model. Let $\mathcal{D}_{3\mu}$ denote the class of read-thrice DNF formulas. We have Theorem 8 to help prove the hardness of learning $\mathcal{D}_{3\mu}$ in the exact model.

The basic idea behind the reduction in the following theorem is this: if the formula g in the proof of Theorem 4 is representable as a $13q$ term DNF formula, then each variable in this representation occurs exactly $13q - 4$ times. Therefore, we generate a set $\{v_1, v_2, \ldots, v_{(13q-4)}\}$ of $13q - 4$ variables for each variable v over which g is defined. We replace each occurrence of variable v with a unique copy, creating a read-once DNF formula. In order to "force the equivalence" of all copies, we add the formula $v_1\overline{v}_2 + v_2\overline{v}_3 + \ldots + v_{13q-4}\overline{v}_1$, for each variable v. Since each variable v_i occurs three times, we have a read-thrice DNF formula. On the other hand, if g cannot be represented as a $13q$-term DNF formula, then at least one variable occurs more than $13q - 4$ times, and hence the new formula cannot be represented as a read-thrice DNF formula. This reduction is similar to the ones in (Angluin & Kharitonov, 1991) and (Aizenstein, Hellerstein & Pitt, 1992). Finally, we show that the class of read-thrice DNF formulas obtained by this reduction is testable with respect to any read-thrice DNF formula.

THEOREM 8 *There exists a class $\mathcal{F}$ such that $\mathcal{F}$ is testable with respect to $\mathcal{D}_{3\mu}$ and REP($\mathcal{D}_{3\mu}$) is NP-Hard for $\mathcal{F}$.*

Proof: The reduction is once again from X3C3. We define $\mathcal{F} = \{g'_{\langle X,C\rangle} : \langle X, C\rangle$ is an instance of X3C3$\}$ as follows. Let $\langle X, C\rangle$ be the given instance of X3C3. Consider the DNF formula g over the set $V = X \cup C$ of $6q$ variables as defined in Theorem 4. We define the DNF formula $g_{3\mu} = g'_{\langle X,C\rangle}$ over a set V' of $O(q^2)$ variables in the following manner.

The set V' contains $13q - 4$ copies of each variable $v \in V$. That is, $V' = \{v_i : v \in V, 1 \le i \le 13q - 4\}$. For all $v \in V$, we let $v_1 = v$. Thus, $V \subset V'$. For the sake of convenience let v_{13q-3} also denote v_1. Since there are $13q - 4$ copies of each variable in V, the number of variables in V' is $6q(13q - 4)$=O(q^2). For any $v \in V$, and integers i and j, $1 \le i, j \le 13q - 4$, we say v_i and v_j are *synonyms*. The DNF formula $g_{3\mu}$ is expressed as the disjunction of terms in $S_1 \cup S_2 \cup S_3$, where S_1 and S_2 are as defined in Theorem 4, and

$$S_3 = \bigcup_{v \in V, 1 \le i \le 13q-4} \{v_i \overline{v}_{i+1}\}.$$

We claim the following:

1. If $g_{3\mu} = g'_{\langle X,C \rangle}$ is any formula in $\mathcal{F}_{6q(13q-4)}$, then $g_{3\mu}$ can be represented as a read-thrice DNF formula if and only if the instance $\langle X, C \rangle$ of X3C3, with $|X| = |C| = 3q$, has an exact cover.

2. $\mathcal{F}$ is testable with respect to $\mathcal{D}_{3\mu}$.

The total number of terms in $S_1 \cup S_2 \cup S_3$ is $15q + (13q - 4)6q$ = O(q^2). Hence this is a polynomial transformation. Therefore, proving items (1) and (2) above would suffice to prove the theorem.

To prove (1), note that by Claims 2 and 3 in Theorem 4, any given instance $\langle X, C \rangle$ of X3C3, with $|X| = |C| = 3q$, has an exact cover if and only if the formula $g = g_{\langle X,C \rangle}$ defined in Theorem 4 can be expressed as a $13q$-term DNF formula over $6q$ variables. Thus, it suffices to prove that $g_{3\mu}$ can be expressed as a read-thrice DNF formula if and only if g can be expressed as a $13q$-term DNF formula.

As a first step, we characterize all the prime implicants of $g_{3\mu}$. The set S_3 expresses the "equivalence" of all synonymous variables. Each term in $S_1 \cup S_2 \cup S_3$ is a prime implicant of $g_{3\mu}$. It is easily shown that for each prime implicant $t \in S_1 \cup S_2 - S_3$, the term t' obtained by replacing any variable $v_i \in t$ (or its complement $\overline{v}_i$) by v_j (correspondingly $\overline{v}_j$), where v_i and v_j are synonyms, is also a prime implicant of $g_{3\mu}$. Denote by S'_1 and S'_2 the set of all prime implicants obtained by such substitutions to prime implicants in S_1 and S_2 respectively. Also, for all pairs v_i and v_j of distinct synonyms, the terms $v_i\overline{v}_j$ and $\overline{v}_i v_j$ are prime implicants of $g_{3\mu}$. Call this set of prime implicants S'_3. To see that the terms in $S' = S'_1 \cup S'_2 \cup S'_3$ are, in fact, the only prime implicants of $g_{3\mu}$, assume that t is any prime implicant of $g_{3\mu}$. If t contains v_i and $\overline{v}_j$ for some $v \in V$ and distinct $1 \le i, j \le 13q - 4$, then $t = v_i\overline{v}_j$ since $v_i\overline{v}_j$ is a prime implicant in S'_3. Otherwise, consider the term t' obtained by replacing each v_i in t, $1 \le i \le 13q - 4$, $v \in V$, with v. The term t' must be a prime implicant of g. Hence, $t' \in S_1 \cup S_2$ and, by construction of S'_1 and S'_2, $t \in S'_1 \cup S'_2$.

Next, we claim the following about g itself.

CLAIM 4 *g can be represented as a DNF formula in which each variable $v \in V$ occurs in exactly* $13q - 4$ *terms if and only if* $\langle X, C \rangle$ *has an exact cover.*

Proof:
$\Leftarrow$: By Claim 2, if $\langle X, C \rangle$ has an exact cover, then g has a minimal DNF representation that contains $13q$ terms. The proof of Claim 2 shows that of these $13q$ terms, $12q$ of them

are the terms in the set S_2 and q terms are from the set S_1. Since each element $x_u \in X$, $1 \leq u \leq 3q$, occurs in exactly 3 sets of C and there are exactly $3q$ sets in C, it is easy to count the number of occurrences of each variable x_u in the terms of set S_2. Each variable x_u occurs (either as a positive literal or as a negative literal) in all terms of S_2 except in 3 terms of the form $\{\mathbf{Z} - \{\overline{c_i}, \overline{x_u}, \overline{x_r}, \overline{x_s}\} \cup \{x_r, x_s\} : c_i \text{ is a set in } C\}$, where $\mathbf{Z} = \{\overline{v} : v \in V\}$. Thus, the terms in S_2 account for $12q - 3$ occurrences of each variable on the form x_u. To count the number of variables of the form x_u in the remaining q terms that express g, note that since $\langle X, C\rangle$ has an exact cover (say C'), each x_u occurs in only one set of C' and is thus, by definition of S_1, missing from only one term. Thus there are $q - 1$ additional occurrences of variable x_u, yielding a total of $13q - 4$ occurrences. To count the number of occurrences of each of the variables of the form $c_i \in V$, $1 \leq i \leq 3q$, note that each such variable occurs in all but 4 terms of S_2, and in all terms of remaining q terms from S_1. Thus, each variable of the form c_i also occurs exactly $13q - 4$ times, if $\langle X, C\rangle$ has an exact cover.

$\Rightarrow$: Note that irrespective of whether or not $\langle X, C\rangle$ has an exact cover, g cannot be expressed with fewer than $13q$ terms. From Claim 3 we can conclude that if $\langle X, C\rangle$ does not have an exact cover, then g requires at least $13q + 1$ terms to be represented as a DNF formula, and must include all terms in S_2. Thus, there are at least $q + 1$ terms from S_1. But this would require more than $13q - 4$ occurrences of each variable of the form c_i, $1 \leq i \leq 3q$. □

CLAIM 5 *$g_{3\mu}$ is representable as a read-thrice DNF formula over V' if and only if g can be represented as a $13q$-term DNF formula over V.*

Proof:
$\Leftarrow$: Let f be a $13q$-term DNF formula that represents g. We can assume that f is a subset of $S_1 \cup S_2$. Consider the DNF formula f' obtained by replacing the i^{th} occurrence of each variable $v \in V$ in f with its synonym v_i. That is,

$$f' = \bigcup_{t \in f} \{v_i : \text{the } i^{th} \text{occurrence of } v \text{ is in } t\}$$

By Claim 4 and the fact that each variable $v \in V$ has $13q - 4$ copies in V', the formula f' is a read-once DNF formula on V'. Now add all terms in S_3 to f'. Since these terms add two more occurrences of each variable in V', f' is now a read-thrice DNF formula. We claim that f' is a DNF representation of $g_{3\mu}$. Clearly $f' \Rightarrow g_{3\mu}$. Further, consider an assignment β over V' such that $g_{3\mu}(\beta) = \mathit{True}$. If β satisfies any term in S_3, then β also satisfies f'. If β falsifies all terms in S_3, then it must satisfy some term in $S_1 \cup S_2$, and hence some term t in f. Moreover, for each variable $v \in V$, we must have $\beta(v_j) = \beta(v)$, for all $1 \leq j \leq 13q - 4$. This implies that the term t' in f' that corresponds to t in f is also satisfied by β. We thus have $g_{3\mu} \Rightarrow f'$.

$\Rightarrow$: Let f' be a read-thrice DNF formula equivalent to $g_{3\mu}$. We can assume that f' is a minimal set of prime implicants of $g_{3\mu}$. For each variable $v \in V$, let $V_v \subset V'$ denote the set of all synonyms of v. We claim that for each $v_i \in V_v$, there exist $v_j, v_k \in V_v$, with distinct i, j and k, such that $v_i\overline{v}_j$ and $\overline{v}_iv_k$ are terms in f'. Consider an assignment β to the variables in V that satisfies exactly five variables in $V - \{v\}$ and falsifies all other variables in V. Such an assignment falsifies g (by the definition of g, no assignment that satisfies more than 4 variables satisfies g), as does the assignment $\beta_{v \leftarrow True}$. Now consider the assignment β' to V' defined as $\beta'(w_u) = \beta(w)$, for all $w \in V$, $1 \leq u \leq 13q - 4$. That is, for each $w \in V$, we assign the value of w to each of the $13q - 4$ variables in V' associated with w. The assignment β' does not satisfy $g_{3\mu}$, since it does not satisfy any term in $S_1 \cup S_2 \cup S_3$. But the assignment $\beta'_{v_i \leftarrow True}$ does. It is easy to see that the only prime implicants in $g_{3\mu}$ that are satisfied by $\beta'_{v_i \leftarrow True}$ are of the form $v_i\overline{v}_j$, and hence there must be at least one such term in f'. Using a similar line of reasoning but starting with the assignment $\gamma = \beta_{v \leftarrow True}$, it can be shown that there must be at least one term of the form $\overline{v}_iv_k$ in f'. Moreover, note that v_j must be distinct from v_k, otherwise the assignment that satisfies all variables except v_i and v_k falsifies f', but not $g_{3\mu}$. Thus, two occurrences of each variable v_i are used in representing prime implicants in S'_3.

Delete all prime implicants of the form $v_i\overline{v}_j$, for all $v \in V$, from f'. The DNF formula f' is read-once and all the prime implicants in it are from $S'_1 \cup S'_2$. We construct a DNF formula f on V as follows: for each term t' in f' add the term t to f, where each literal in t' that corresponds to a variable v_i in V' is replaced with the corresponding literal from variable v in V. Thus, there are at most $13q - 4$ occurrences of each variable $v \in V$ in f. We claim that f is a DNF representation of g. By the definition of S'_1 and S'_2, each term in f is a prime implicant of g, and hence $f \Rightarrow g$. Now consider any assignment β that satisfies g. Extend β to β' over the set V' by setting $\beta'(v_i) = \beta(v)$, for all $v \in V$, $1 \leq i \leq 13q - 4$. Since β satisfies some term in $S_1 \cup S_2$, β' satisfies $g_{3\mu}$. Thus, β' satisfies some term t in $f' \equiv g_{3\mu}$ such that $t \in S'_1 \cup S'_2$. By our construction of f, this implies that some term in f is satisfied by β itself, since all synonyms are assigned the same value. Hence, $g \Rightarrow f$. The claim then follows from Claims 2, 3 and 4. □

We now show that (2) is satisfied by $\mathcal{F}$—that is, $\mathcal{F}$ is testable with respect to read-thrice DNF formulas. To do this, we construct an algorithm, $\mathcal{A}_{3\mu}$, such that $\mathcal{A}_{3\mu}$ takes as input a read-thrice DNF formula h over $6q(13q-4)$ variables and $g_{3\mu} \in \mathcal{F}_{6q(13q-4)}$ and outputs in time polynomial in $|h|$, $|g|$ and $|q|$ either (i) "No" and an assignment x such that $g(x) \neq h(x)$ or (ii) "Yes" and a read-thrice DNF formula h' over $6q(13q-4)$ variables such that $h' \equiv g$.

Let **True** denote the assignment that sets all variables in V' to *True*. For any $v \in V$, let $\mathbf{True}^v$ denote the assignment over V' that sets all synonyms of v to *False* and the remaining variables in V' to *True*. That is, $\mathbf{True}^v(v_i) = False$, for all $1 \leq i \leq 13q - 4$, and $\mathbf{True}^v(u) = True$ for all $u \in V' - \{v_1, v_2, ..., v_{13q-4}\}$.

Algorithm $\mathcal{A}_{3\mu}$ creates a new formula h' from h as follows. For each term t in h: (i) if t is of the form $v_i\overline{v}_jz$, where v_i and v_j are distinct synonyms of some $v \in V$, and z is some set of literals, $\mathcal{A}_{3\mu}$ replaces t with $v_i\overline{v}_j$; (ii) if t is of the form $v_{i_1}v_{i_2}...v_{i_k}z$, where $v_{i_1}, v_{i_2}, ..., v_{i_k}$ are synonyms of some $v \in V$, and z contains no synonyms of v, $\mathcal{A}_{3\mu}$ replaces t with $v_{i_1}z$. (iii) if t is of the form $\overline{v}_{i_1}\overline{v}_{i_2}...\overline{v}_{i_k}z$, where $v_{i_1}, v_{i_2}, ..., v_{i_k}$ are synonyms of some $v \in V$, and z contains no synonyms of v, $\mathcal{A}_{3\mu}$ replaces t with $v_{i_1}z$. All other terms are retained.

If $h'(\mathbf{True}) = True$, then $\mathcal{A}_{3\mu}$ outputs "No" and **True**. If there exists a $v \in V$ such that $h'(\mathbf{True}^v) = True$, then $\mathcal{A}_{3\mu}$ outputs "No" and $\mathbf{True}^v$. If there exists a $v \in V$

and an i, $1 \leq i \leq 13q - 4$, such that $h'(\mathbf{True}_{v_i \leftarrow False}) = False$, then $\mathcal{A}_{3\mu}$ outputs "No" and $\mathbf{True}_{v_i \leftarrow False}$. If there exists a $v \in V$ and an i, $1 \leq i \leq 13q - 4$, such that $h'(\mathbf{True}^v_{v_i \leftarrow True}) = False$, then $\mathcal{A}_{3\mu}$ outputs "No" and $\mathbf{True}^v_{v_i \leftarrow True}$. Otherwise, $\mathcal{A}_{3\mu}$ creates h'' from h' by replacing every synonym v_i, $1 \leq i \leq 13q - 4$, of every variable $v \in V$ with v, and then deleting all terms of the form $v\overline{v}$. $\mathcal{A}_{3\mu}$ then invokes the equivalence testing algorithm $\mathcal{A}$ in the proof of Theorem 4 with h'' and g as inputs. If $\mathcal{A}$ outputs "No" and an assignment β over V, then $\mathcal{A}_{3\mu}$ outputs "No" and the assignment β' over V' formed by replicating the assignment to each $v \in V$ to all its synonyms. That is, $\beta'(v_i) = \beta(v)$ for all $v \in V$ and $1 \leq i \leq 13q - 4$. If $\mathcal{A}$ outputs "Yes" then $\mathcal{A}_{3\mu}$ outputs "Yes" and the DNF formula h' modified as follows: delete from h' all terms of the form $v_i\overline{v}_j$ for some v, i and j, and add all terms in S_3.

We now show that $\mathcal{A}_{3\mu}$ is correct. Note that $g_{3\mu}(\mathbf{True}) = g_{3\mu}(\mathbf{True}^v) = False$ for all $v \in V$. Also note that for any assignment β returned by $\mathcal{A}_{3\mu}$ such that $h'(\beta) \neq g_{3\mu}(\beta)$ it also holds that $h(\beta) \neq g_{3\mu}(\beta)$. As a first step we show that if $\mathcal{A}_{3\mu}$ invokes $\mathcal{A}$, then for each $v \in V$ and $1 \leq i \leq 13q - 4$, there exist $1 \leq j, k \leq 13q - 4$ (both j and k distinct from i) such that $v_i\overline{v}_j$ and $\overline{v}_i v_k$ are terms in h'. Assume to the contrary that there exists some v_i such that there is no term of the form $v_i\overline{v}_j$ in h'. Since, at this point, $h'(\mathbf{True}^v) = False$ and $h'(\mathbf{True}^v_{v_i \leftarrow True}) = True$, there must be a term t in h' of the form $v_i z$, where z is a set of positive literals that do not correspond to synonyms of v. But such a term (and hence h') would be satisfied by $\mathbf{True}$, which is a contradiction. Now assume that there is no term of the form $\overline{v}_i v_k$ in h'. Again, we know that $h'(\mathbf{True}) = False$ and $h'(\mathbf{True}_{v_i \leftarrow False}) = True$. Therefore, there must be a term t in h' of the form $\overline{v}_i z$, where z is a set of positive literals that do not correspond to synonyms of v. But such a term (and hence h') would be satisfied by $\mathbf{True}^v$, which is a contradiction.

Thus, when h'' is created, h' contains at most one occurrence of each synonym variable v_i in a term that is not of the form $v_i\overline{v}_j$ or $\overline{v}_i v_k$. There are, therefore, at most $13q - 4$ occurrences of each variable $v \in V$ in h''. Clearly, if $\mathcal{A}$ responds with a "Yes", $\mathcal{A}_{3\mu}$ outputs a read-thrice DNF formula equivalent to $g_{3\mu}$. If $\mathcal{A}$ responds with an assignment β such that $h''(\beta) \neq g(\beta)$, then it is easy to verify that the assignment β' satisfies $h'(\beta) \neq g_{3\mu}(\beta')$.

Finally, note that all steps in $\mathcal{A}_{3\mu}$ can be carried out in time polynomial in $|h|$, and q.

■

COROLLARY 8 *If P $\neq$ NP, then the class of read-thrice DNF formulas is not properly learnable in the exact model.*

Proof: Follows directly from Theorem 8 and Corollary 1. □

Next we consider the learnability of read-thrice DNF formulas in the extended PAC model. We have the following theorem.

THEOREM 9 *There exists a class $\mathcal{F}$ of Boolean formulas compressible with respect to $\mathcal{D}_{3\mu}$ such that REP($\mathcal{D}_{3\mu}$) is NP-Hard for $\mathcal{F}$.*

Proof: We show that the class $\mathcal{F}$ as defined in the preceding theorem is compressible with respect to read-thrice DNF formulas. This, in conjunction with the fact that REP($\mathcal{D}_{3\mu}$) is NP-hard for $\mathcal{F}$, proves the theorem.

CLAIM 6 *$\mathcal{F}$ is compressible with respect to $\mathcal{D}_{3\mu}$.*

Proof: To show that $\mathcal{F}$ is compressible, we give an algorithm $\mathcal{A}$ such that if $\mathcal{A}$ is given as input a formula $g_{3\mu} = g'_{\langle X,C\rangle} \in \mathcal{F}_{6q(13q-4)}$, $\mathcal{A}$ outputs in time polynomial in q a set $X_{g_{3\mu}}$ of assignments such that if there exists a read-thrice DNF formula f over $6q(13q-4)$ variables that is consistent with $g_{3\mu}$ over all assignments in $X_{g_{3\mu}}$, then there exists a read-thrice DNF formula f' over $6q(13q-4)$ variables such that $g_{3\mu} \equiv f'$.

Consider the set X_g of assignments over V defined in the proof of Theorem 6. Extend each assignment in X_g to the set V' by replicating the assignment to all synonyms. That is, $X'_g = \{\beta' : \beta'(v_i) = \beta(v), \beta \in X_g, v \in V, 1 \le i \le 13q-4\}$. As in Theorem 8, let **True** be the assignment that satisfies all variables in V', and for any $v \in V$, let $\mathbf{True}^v$ denote the assignment over V' that sets all synonyms of v to *False* and the remaining variables in V' to *True*. Let $P' = \{\mathbf{True}_{v_i \leftarrow False} : v \in V, 1 \le i \le 13q-4\}$. For all $v \in V$, let $Q^v = \{\mathbf{True}^v_{v_i \leftarrow True} : 1 \le i \le 13q-4\}$. Finally, we define $X_{g'_{\langle X,C\rangle}} = X_{g_{3\mu}} = \{\mathbf{True}\} \cup X'_g \cup P' \cup \bigcup_{v \in V} Q^v$. The number of assignments in X'_g is at most $(30q+1)(6q+1)$ (from the proof of Theorem 6, the number of assignments in P' is $(6q)(13q-4)$, the number of assignments in $\bigcup_{v\in V} Q^v$ is $(6q)(13q-4)$. Thus, the total number of assignments in $X_{g_{3\mu}}$ is at most $O(q^2)$.

We now show that there exists a read-thrice DNF formula f over V' that is consistent with $g_{3\mu}$ over the evaluation of assignments in $X_{g_{3\mu}}$ only if $g_{3\mu}$ can be represented as a read-thrice DNF formula.

Let f be a read-thrice DNF formula consistent with $g_{3\mu}$ over the evaluation of assignments in $X_{g_{3\mu}}$. We create a read-thrice DNF formula f' with no more terms than f as follows: For each term t in f of the form $v_i\overline{v}_j z$, where v_i and v_j are synonyms and z is any conjunction of literals, add $v_i\overline{v}_j$ to f'. The formula thus obtained is still consistent with $g_{3\mu}$ over the evaluation of all assignments in $X_{g_{3\mu}}$. Next, for each term in f of the form v_iv_jz (or correspondingly $\overline{v}_i\overline{v}_jz$), where v_i and v_j are synonyms and z is any conjunction of literals, add v_iz ($\overline{v}_iz$) to f'. The formula thus obtained is consistent with $g_{3\mu}$ over all assignments in $X_{g_{3\mu}}$. Finally, delete all terms in f' that are not satisfied by any assignment in $X_{g_{3\mu}}$ that satisfies $g_{3\mu}$.

The argument in the proof of Theorem 8 that shows $\mathcal{F}$ is testable with respect to read-thrice DNF formulas can be repeated to show that for each variable $v_i \in V'$, there exist v_j and v_k such that $v_i\overline{v}_j$ and $\overline{v}_iv_k$ are terms in f', lest it not be consistent with $g_{3\mu}$ over $X_{g_{3\mu}}$. Thus, there is only one remaining occurrence of each variable in V'. Now, delete all two literal terms in f' and replace them with the terms in S_3. The formula f' thus obtained is still consistent with $X_{g_{3\mu}}$. If there exists a term t in f' of length greater than 2 that is not a superset of any of the terms in $S'_1 \cup S'_2$, then clearly f was not consistent with X'_g. Replacing each term t in f' with a subset t' that is a prime implicant of $g_{3\mu}$ and arguing on the lines of Theorem 6 shows that f' is a read-thrice DNF formula equivalent to $g_{3\mu}$. ■

COROLLARY 9 *If RP $\neq$ NP, then the class of read-thrice DNF formulas is not properly learnable in the extended PAC model.*

Proof: Follows directly from Theorem 9 and Corollary 2. □

7. Conclusions

We have presented general techniques to prove that certain classes of Boolean formulas cannot be efficiently learned when an exact (or PAC) learning algorithm is restricted to hypotheses from the class itself. We have applied these techniques to show that n^α-term DNF formulas over n Boolean variables, for any fixed constant $\alpha > 0$, cannot be properly learned in the exact model unless P = NP and in the extended PAC model unless RP = NP. Using a very strong assumption about the containment of NP in DTIME classes, we also show that there exists a fixed constant $c > 0$ such that $\log^c n$-term DNF formulas cannot be learned properly in the exact model. Finally, we have improved the result in (Aizenstein, Hellerstein & Pitt, 1992) to show that read-thrice DNF formulas are not learnable properly in the exact model unless P = NP and in the extended PAC model unless RP = NP.

One common feature of this technique and the techniques in (Pitt & Valiant, 1988), (Kearns, et al., 1987) and (Aizenstein, Hellerstein & Pitt, 1992) is that all of them rely solely on the apparent hardness of finding a representation in a concept class that agrees with given data. These techniques do not exploit the fact that a learning algorithm has no knowledge of the data at all and must somehow elicit it using queries. On the other hand, Angluin's technique of approximate fingerprints (Angluin, 1989) relies solely on this "lack of knowledge" to devise an adversary that will confound any purported efficient learning algorithm that uses only equivalence queries. Recently, this technique was generalized to give a characterization of polynomial-query learnability with equivalence and membership queries (Hellerstein, et al., 1995). However, Angluin's technique and its generalization do not exploit the limited computational resources—in particular time—available to the learning algorithm. Perhaps there is a way to combine these disparate techniques to form a stronger technique to attack the problem of learning DNF formulas in the normal sense.

References

Angluin, D., Hellerstein, L. & Karpinski, M. (1993). Learning Read-Once Formulas with Queries. *Journal of the ACM*, pages 185–210.

Aizenstein, H., Hellerstein, L. & Pitt, L. (1992). Read-Thrice DNF is Hard to Learn With Membership and Equivalence Queries. *Proceedings of the 33rd Annual IEEE Symposium on the Foundations of Computer Science*, pages 523–532.

Aizenstein, H. (1993). *On the Learnability of Disjunctive Normal Form Formulas and Decision Trees*. PhD thesis, University Of Illinois at Urbana-Champaign.

Angluin, D. & Kharitonov, M. (1991). When Won't Membership Queries Help? *Proceedings of the ACM Symposium on Theory of Computing*, pages 444–454.

Angluin, D. (1988). Queries and Concept Learning. *Machine Learning*, 2:319–342, 1988.

Angluin, D. (1989). Equivalence Queries and Approximate Fingerprints. *Proceedings of the Second Annual Workshop on Computational Learning Theory*, pages 134–145.

Angluin, D. (1990). Negative Results for Equivalence Queries. *Machine Learning*, 5:121–150.

Aizenstein, H. & Pitt, L. (1991). Exact Learning of Read-twice DNF Formulas. *Proceedings of the 32nd Annual IEEE Symposium on the Foundations of Computer Science*, pages 170–179.

Aizenstein, H. & Pitt, L. (1992). Exact Learning of Read-k Disjoint DNF and Not-So-Disjoint DNF. *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 71–76.

Berggren, U. (1993). Linear Time Deterministic Learning of k-term DNF. *Proceedings of the Sixth Annual Workshop on Computational Learning Theory*, pages 37–40.

Blum, A. & Rudich, S. (1992). Fast Learning of k-term DNF Formulas with Queries. *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 382–389.

Bshouty, N., Cleve, R., Kannan, S. & Tamon, C. (1994). Oracles and Queries that are Sufficient for Exact Learning. *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pages 130–139.

Bshouty, N., Goldman, S., Hancock, T. & Matar, S. (1993). Asking Questions to Minimize Errors. *Proceedings of the 6th Annual ACM Conference on Computational Learning Theory*, pages 41–50.

Garey, M. & Johnson, D. (1979). *Computers and Intractability*. W.H. Freeman and Company, New York.

Hancock, T. (1991). Learning 2μDNF Formulas and $k\mu$ Decision Trees. *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 199–209.

Hancock, T. 1992. *The Complexity of Learning Formulas and Decision Trees that Have Restricted Reads*. PhD thesis, Harvard University, Center for Research in Computing Technology, Aiken Computation Laboratory. TR-15-92.

Hellerstein, L., Pillaipakkamnatt, K., Raghavan, V. & Wilkins, D. (1995). How Many Queries are Needed to Learn? *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing*.

Jackson, J. (1994). An Efficient Membership-Query Algorithm for Learning DNF with Respect to the Uniform Distribution. *Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science*.

Kearns, M., Li, M., Pitt, L. & Valiant, L. (1987). On the Learnability of Boolean Formulae. *Proceedings of the ACM Symposium on Theory of Computing*, pages 285–295.

Pillaipakkamnatt, K. (1995). *Proper Learnability of Boolean Formulas in Disjunctive Normal Form*. PhD thesis, Vanderbilt University.

Pillaipakkamnatt, K. & Raghavan, V. (1994). X3C3 is NP-Complete. Technical Report TR-94-62, Department of Computer Science, Vanderbilt University, Nashville, TN.

Pillaipakkamnatt, K. & Raghavan, V. (1995). Read-Twice DNF Formulas are Properly Learnable. *Information and Computation (to appear)*.

Pitt, L. & Valiant, L. (1988). Computational Limitations on Learning from Examples. *Journal of the ACM*, 35(4):965–984.

Valiant, L. (1984). A Theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Received July 5, 1994
Accepted February 10, 1995
Final Manuscript March 1, 1995