Tight Bounds on Proper Equivalence Query Learning of DNF

Lisa Hellerstein HSTEIN@POLY.EDU
Devorah Kletenik DKLETENIK@CIS.POLY.EDU
Linda Sellie SELLIE@MAC.COM
Polytechnic Institute of NYU

Rocco Servedio ROCCO@CS.COLUMBIA.EDU
Columbia University

Editor: Shie Mannor, Nathan Srebro, Robert C. Williamson

Abstract

We prove a new structural lemma for partial Boolean functions f, which we call the *seed lemma for DNF*. Using the lemma, we give the first subexponential algorithm for proper learning of poly(n)-term DNF in Angluin's Equivalence Query (EQ) model. The algorithm has time and query complexity $2^{(\tilde{O}\sqrt{n})}$, which is optimal. We also give a new result on certificates for DNF-size, a simple algorithm for properly PAC-learning DNF, and new results on EQ-learning $\log n$-term DNF and decision trees.

Keywords: query learning, exact learning, proper learning, DNF, certificates

1. Introduction

Over twenty years ago, Angluin (1988, 1990) began study of the equivalence query (EQ) learning model. Valiant (1984) had asked whether n-variable DNF formulas with poly(n) terms were poly(n)-time learnable in the PAC model; this question is still open. Angluin asked the same question in the EQ model. She introduced the method of "approximate fingerprints" and used it to prove that any *proper* algorithm for EQ-learning poly(n)-term DNF formulas requires super-polynomial query complexity, and hence super-polynomial time. (In a proper DNF learning algorithm, all hypotheses are DNF formulas.)

Angluin's work left open the problem of determining the exact complexity of EQ-learning poly(n)-term DNF, both properly and improperly. Tarui and Tsukiji (1999) noted that Angluin's proof can be modified to show that a proper EQ algorithm must have query complexity at least $2^{\tilde{O}(\sqrt{n})}$. (They did not give details, but we shall prove this explicitly as a consequence of a more general result.) The fastest *improper* algorithm for EQ-learning DNF is due to Klivans and Servedio (2004, Corollary 12), and runs in time $2^{\tilde{O}(n^{1/3})}$.

Our positive results. In this paper, we give the first subexponential-time algorithm for *proper* learning of poly(n)-term DNF in the EQ model. Our algorithm has time and query complexity that, like the lower bound, is $2^{\tilde{O}(\sqrt{n})}$.

Our EQ algorithm implies a new result on certificates for DNF size. Hellerstein et al. (1996) asked whether DNF has "poly-size certificates," that is, whether there are polynomials q and r such that for all $s, n > 0$, n-variable functions requiring DNF formulas of size greater than $q(s, n)$ have certificates of size $r(s, n)$ certifying that they do not have DNF formulas of size at most s. (By the results of Hellerstein et al. (1996), this is equivalent to asking whether DNF can be properly learned using polynomially many membership and equivalence queries, with hypotheses of polynomial size.) Our result does not resolve this question, but it shows that there are analogous subexponential certificates. More specifically, it shows that there exists a function $r(s,n) = 2^{O(\sqrt{n \log s} \log n)}$ such that for all $s, n > 0$, n-variable functions requiring DNF formulas of size greater than $r(s,n)$ have certificates of size at most $r(s,n)$ certifying that they do not have DNF formulas of size at most s.

Our EQ algorithm is based on a new structural lemma for partial Boolean functions, which we call the *seed lemma for DNF*. It states that if a partial Boolean function f has at least one positive example and is consistent with a DNF of size s, then f has a projection f_p, induced by fixing the values of $O(\sqrt{n \log s})$ variables, such that f_p has at least one positive example, and is consistent with a conjunction of literals.

We also use the seed lemma for DNF to obtain a new subexponential proper algorithm for PAC-learning DNFs which is simpler than the previous algorithm of Alekhnovich et al. (2009), with the same bounds. The earlier algorithm runs multiple recursive calls in round robin fashion until one succeeds; in contrast, our algorithm is iterative with a simple analysis.

Decision-trees can be PAC and EQ-learned in time $n^{O(\log s)}$, where s is the size of the tree (Ehrenfeucht and Haussler, 1989; Simon, 1995). We prove a seed lemma for decision trees as well, and use it to obtain an algorithm that learns decision trees using DNF hypotheses in time $n^{O(\log s_1)}$, where s_1 is the number of 1-leaves in the tree. (For any "minimal" tree, the number of 0-leaves is at most ns_1; this bound is tight for the optimal tree computing a conjunction of n literals.)

Our negative results. We prove a lower bound result that quantifies the tradeoff between the number of queries needed to properly EQ-learn DNF formulas, and the size of such queries. One consequence is a lower bound of $2^{\Omega(\sqrt{n \log n})}$ on the query complexity necessary for an EQ algorithm to learn DNF formulas of size poly(n), using DNF hypotheses. This matches the lower bound of $2^{(\tilde{O}\sqrt{n})}$ mentioned by Tarui and Tsukuji. The bound for our EQ algorithm, applied to DNF formulas of size $\text{poly}(n)$, differs from this lower bound by only a factor of $\log n$ in the exponent.

We also prove a negative result on learning $\log n$-term DNF using DNF hypotheses. Several $\text{poly}(n)$-time algorithms are known for this problem in the membership and equivlence query (MEQ) model (Bshouty, 1997; Blum and Rudich, 1995; Bshouty et al., 1996b; Hellerstein and Raghavan, 2005). We prove that the membership queries are essential, by showing that there is no $\text{poly}(n)$-time algorithm that learns $O(\log n)$-term DNF using DNF hypotheses with equivalence queries alone. In contrast, Angluin and Kharitonov (1995) showed that, under cryptographic assumptions, membership queries do not help in PAC-learning unrestricted DNF formulas. Blum and Singh (1990) gave an algorithm that PAC-learns $\log n$-term DNF using DNF hypotheses of size $n^{O(\log n)}$ in time $n^{O(\log n)}$; our results imply that significantly better bounds than these cannot be achieved in the equivalence query model.

2. Preliminaries

A *literal* is a variable or its negation. A *term*, also called a *monomial*, is a possibly empty conjunction ($\wedge$) of literals. If the term is empty, all assignments satisfy it. The *size* of a term is the number of literals in it. We say that term t *covers* assignment x if $t(x) = 1$. It is an implicant of Boolean function $f(x_1, \ldots, x_n)$ if $t(x) = 1$ implies $f(x) = 1$. A *DNF* (disjunctive normal form) formula is either the constant 0, the constant 1, or a formula of the form $t_1 \vee \cdots \vee t_k$, where $k \geq 1$ and each t_i is a term. A k-term DNF is a DNF formula consisting of at most k terms. A k-DNF is a DNF formula where each term has size at most k. The *size* of a DNF formula is the number of its terms; if it is the constant 0 or 1, its size is 1.

A *partial* Boolean function f maps $\{0,1\}^n$ to $\{0,1,*\}$, where $*$ means undefined. A Boolean formula ϕ is *consistent* with a partial function f (and vice versa) if $\phi(x) = f(x)$ for all $x \in \{0,1\}^n$ where $f(x) \neq *$. If f is a partial function, then $\mathit{dnf\text{-}size}(f)$ is the size of the smallest DNF formula consistent with f. Assignment $x \in \{0,1\}^n$ is a *positive example* of (partial or total) Boolean function $f(x_1,\ldots,x_n)$ if $f(x)=1$, and a *negative example* if $f(x)=0$. A *sample* of f is a set of pairs $(x, f(x))$, where $x \in \{0,1\}^n$.

Let $X_n = \{x_1,\ldots,x_n\}$. A *projection* of a (partial) function $f(x_1,\ldots,x_n)$ is a function induced from f by fixing k variables of f to constants in $\{0,1\}$, where $0 \leq k \leq n$. We consider the domain of the projection to be the set of assignments to the remaining $n-k$ variables. If T is a subset of literals over X_n, or a term over X_n, then f_T denotes the projection of f induced by setting the literals in T to 1.

A *certificate* that a property P holds for a Boolean function $f(x_1,\ldots,x_n)$ is a set $A \subseteq \{0,1\}^n$ such that for all Boolean functions $g(x_1,\ldots,x_n)$, if g does not have property P, then $f(a) \neq g(a)$ for some $a \in A$. The *size* of certificate A is $|A|$.

For $x \in \{0,1\}^n$ we write $|x|$ to denote $\sum_i x_i$ and $\mathsf{Maj}(x_1,\ldots,x_n)$ to denote the majority function whose value is 1 if $\sum_{i=1}^n x_i \geq n/2$ and 0 otherwise. We write "log" to denote log base 2, and $\tilde{O}(\cdot)$ to indicate that we are supressing factors that are logarithmic in the arguments to $\tilde{O}()$. Finally, we use standard models and definitions from computational learning theory. We omit these here; more information can be found in Appendix A.

3. Seeds

Definition 1 *A* seed *of a partial Boolean function $f(x_1,\ldots,x_n)$ is a (possibly empty) monomial T that covers at least one positive example of f, such that f_T is consistent with a monomial.*

Lemma 2 below is our main new structural lemma. At a high level its proof is similar to proofs of earlier results in proof complexity (see e.g. Ben-Sasson and Wigderson (2001)) and DNF-learning (Bshouty, 1996; Alekhnovich et al., 2004) but there are some significant differences. In these earlier results the goal was to construct a projection of a given DNF causing all the terms of the reduced DNF to be short, or forcing the DNF to be identically 0 or identically 1. Here we construct a projection that reduces the DNF to a single monomial, and the reduced DNF cannot be identically 0. In addition, our lemma applies to partial functions and not just to total functions.

Lemma 2 (Seed lemma for DNF) *Let f be a partial Boolean function such that $f(a)=1$ for some $a \in \{0,1\}^n$. Let $s = \mathit{dnf\text{-}size}(f)$. Then f has a seed of size at most $3\sqrt{n \ln s}$.*

Proof Let ϕ be a DNF formula of size $s = \mathit{dnf\text{-}size}(f)$ that is consistent with f. If $s=1$, then the empty monomial $\emptyset$ is a seed. Suppose $s>1$. Then ϕ must have at least two terms. Since ϕ has size $s = \mathit{dnf\text{-}size}(f)$, it is of minimum size, so each term of ϕ covers at least one positive example of f. We construct a seed T from ϕ by iteratively selecting a literal in ϕ, and either replacing all occurences of that literal by 0 or by 1. (We also replace any occurences of the negation of that literal by the complementary constant.) These replacements correspond to a projection of the partial function f. When we select a literal of ϕ to set to 0, we do so in order to remove the terms containing the literal from ϕ. We store such selected literals in a set Z. When we select a literal to set to 1, we do so because it appears in all remaining terms of ϕ, and setting it to 1 reduces the length of all of those terms. We store these literals in a set R. During the process, we also remove terms from ϕ if they no longer cover any positive examples of the current projection of partial function f (i.e. if they only cover $*$'s).

More specifically, we initalize the two sets Z and R to be empty, and then repeat the following steps until a seed is output:

1. If there is a term P of ϕ of size at most $\sqrt{n \ln s}$, output the conjunction of the literals in $Z \bigcup P'$ as a seed, where P' is the set of literals in P. (Note that P could be the empty monomial, which is satisfied by all assignments.)

2. If all terms of ϕ have size greater than $\sqrt{n \ln s}$, check whether there is a literal l whose value is not fixed by the projection $f_{Z \bigcup R}$, such that l is satisfied by all positive examples of $f_{Z \bigcup R}$ and:

   (a) If so, add l to R. Set l to 1 in ϕ by removing all occurrences of l in the terms of ϕ. (There are no occurrences of $\bar{l}$ in ϕ.)

   (b) If not, let l be the literal appearing in the largest number of terms of ϕ. Add $\bar{l}$ to Z. Set l to 0 in ϕ by removing from ϕ all terms containing l, and removing all occurrences of $\bar{l}$ in the remaining terms. Then remove any terms which do not cover a positive example of $f_{Z \cup R}$.

We now prove that the above procedure outputs a seed with the desired properties. Initially, ϕ is a DNF of minimum size, so each term of ϕ covers at least one positive example of f. During execution of Step 2a, no terms are deleted. If Step 2b is executed instead of 2a, there is a term t of ϕ that does not contain the l chosen in 2b, and hence t is not deleted in that step. Literals are only added to R in Step 2a, when there is a literal l satisfied by all positive examples of $f_{Z \bigcup R}$.

Using the above observations, it is easy to show that the following four invariants are maintained by the procedure: (1) ϕ is consistent with $f_{Z \bigcup R}$, (2) ϕ contains at least one (possibly empty) term, (3) each term of ϕ covers at least one positive example of $f_{Z \bigcup R}$ (which is why, in Step 2a, no term contains $\bar{l}$), and (4) for any positive example a of f, if a satisfies all literals in Z, then a satisfies all literals in R.

We now show that the output set, $Z \bigcup P'$, is a seed. By the invariants, ϕ is consistent with $f_{Z \bigcup R}$, and term P of ϕ is satisfied by at least one positive example of $f_{Z \bigcup R}$. Thus $f_{Z \bigcup P'}$ has at least one positive example. Further, since P is a term of ϕ, and ϕ is consistent with $f_{Z \bigcup R}$, if an assignment a satisfies $Z \bigcup P' \bigcup R$ then $f(a) = 1$ or $f(a) = *$. Thus $f_{Z \bigcup P'}$ is consistent with the monomial $\bigwedge_{l \in R} l$, and $Z \bigcup P'$ is a seed.

It remains to show that the output seed has size at most $3\sqrt{n \ln s}$. By construction, P has at most $\sqrt{n \ln s}$ literals. To bound the size of Z we use a standard technique (cf. Angluin (1990)). Each time a literal is added to Z, all terms of ϕ have size at least $\sqrt{n \ln s}$, thus of the $2n$ possible literals, the one occuring most frequently appears in at least a fraction α of the terms, where $\alpha = \frac{1}{2}\sqrt{(\ln s)/n}$. So each time a literal is added to Z, the fraction of terms removed from ϕ is at least α. When Z contains r literals, ϕ contains at most $(1-\alpha)^r s$ terms. For $r \geq 2\sqrt{n \ln s}$, $(1-\alpha)^r s < e^{-\alpha r} s \leq 1$. Since ϕ always contains at least one term, Z contains at most $2\sqrt{n \ln s}$ literals. Thus T has size at most $3\sqrt{n \ln s}$. ■

The bound of Lemma 2 on seed size is nearly tight for a monotone DNF formula on n variables having $\sqrt{n}$ disjoint terms, each of size $\sqrt{n}$. The smallest seed for the function it represents has size $\sqrt{n}-1$.

4. PAC-learning DNF (and decision trees) using seeds

We begin by presenting our algorithm for PAC-learning DNFs. It is simpler than our EQ algorithm, and the ideas used here are helpful in understanding that algorithm. We present only the portion of the PAC algorithm that constructs the hypothesis from an input sample S, and we assume that the size s of the target DNF formula is known. The rest of the algorithm and analysis is routine (see e.g. (Alekhnovich et al., 2009)).

Our algorithm takes as input a sample S of a Boolean function $f : \{0,1\}^n \to \{0,1\}$. It outputs a DNF formula ϕ consistent with f on S (i.e. such that $\phi(x) = f(x)$ for all $(x, f(x)) \in S$). We describe the algorithm here and give the pseudocode in Appendix B.

The algorithm begins with a hypothesis DNF h that is initialized to 0. It finds terms one by one and adds them to h. Each additional term covers at least one uncovered positive example in S, and terms are added to h until all positive examples in S are covered.

We now describe the procedure for finding a term. We write f^S to denote the partial Boolean function defined on $\{0,1\}^n$ such that $f^S(x) = f(x)$ if x is such that $(x, f(x)) \in S$, and $f^S(x) = *$ otherwise.

First, the algorithm tests each conjunction T of size at most $3\sqrt{n \ln s}$ to determine whether it is a seed of f^S. To perform this test, the algorithm first explicitly checks whether T covers at least one positive example in S; if not, T is not a seed. If so, the algorithm checks whether f^S_T is consistent with a monomial as follows. Let S_T denote the set of positive examples in S that satisfy T. The algorithm computes term T', which is the conjunction of the literals that are satisfied by all assignments in S_T (so $T \subseteq T'$). One can show that f^S_T is consistent with a monomial iff all negative examples in S falsify T'. So, the algorithm checks whether all negative examples in S falsify T'. (In effect, the algorithm runs the standard PAC algorithm for learning monomials on the sample consisting of S_T and all negative examples in S, yielding term T', and then it checks whether T' is really consistent with that sample.)

By the seed lemma for DNF, at least one seed T will be found. For each seed T found, the associated term T' is added to h, and the positive examples satisfying T' are removed from S. If S still contains a positive example, the procedure is repeated with the new S. *Note that a conjunction T that is not a seed of f^S at the start of the algorithm may become a seed later, after positive examples have been removed from S.*

The correctness of the algorithm follows immediately from the above discussion. Once a seed T is found, all positive examples in S that satisfy T are removed from S, and thus the same seed will never be found twice. Thus the algorithm runs in time $2^{O(\sqrt{n \log s} \log n)}$ and outputs a DNF formula of that size.

Connection to prior work. Let us say that an algorithm uses the *seed covering method* if it builds a hypothesis DNF from an input sample S by repeatedly executing the following steps, until no positive examples remain in the sample: (1) find a seed T of partial function f^S, (2) form a term T' from the positive examples in S that satisfy T, by taking the conjunction of the literals satisfied by all those examples, (3) add term T' to the hypothesis DNF and remove from S all positive examples covered by T'.

With this perspective, the algorithm of Blum and Singh (1990), which PAC-learns k-term DNF, implicitly uses the seed covering method. It first finds seeds of size $k-1$, then size $k-2$, and so on. It differs from our DNF-learning algorithm in that it only searches for a restricted type of seed. Our seeds are constructed from two types of literals, those (in Z) that eliminate terms from the target, and those (in P) that satisfy a term. Their algorithm only searches for seeds containing the first type of literal. Their algorithm works by identifying subsets of examples satisfying the same subset of terms of the target, while ours works by identifying subsets of examples satisfying a common term of the target.

Learning decision trees. We conclude this section by observing that the seed method can also be used to learn decision trees in time $n^{O(\log s_1)}$, where s_1 is the number of 1-leaves in the decision tree. This follows easily from the following lemma.

Lemma 3 *(Seed lemma for decision trees) Let f be a partial Boolean function, such that f has at least one positive example, and f is consistent with a decision tree having s_1 leaves that are labeled 1. Then f has a seed of size at most* $\log s_1$.

Proof Let J be a decision tree consistent with f that has s_1 leaves labeled 1. Without loss of generality, assume that each 1-leaf of J is reached by at least one positive example of f. Define an internal node of J to be a *key* node if neither of its children is a leaf labeled 0. Define the *key-depth* of a leaf to be the number of key nodes on the path from the root down to it. It is not hard to show that since J has s_1 leaves labeled 1, it must have a 1-leaf with key-depth at most $\log s_1$. Let p be the path from the root to this 1-leaf. Let L be the set of literals that are satisfied along path p. Let Q be the conjunction of literals in L that come from key nodes, and let R be the conjunction of the remaining literals. Consider an example x that satisfies Q. Consider its path in J. If x also satisfies R, it will end in the 1-leaf at the end of p, else it will diverge from p at a non-key node, ending at the 0-child of that node. Thus f_Q is consistent with monomial R, Q is a seed of f, and $|Q| \leq \log s_1$. ■

5. EQ-learning DNF using seeds

We now describe our algorithm for EQ-learning DNF. It can be viewed as learning a decision list with monomials of bounded size in the nodes, and (implicant) monomials of unbounded size in the leaves (and 0 as the default output bit); we use a variant of the (Helmbold et al., 1990; Simon, 1995) approach to EQ-learn decision lists with bounded-size monomials in the nodes and constant output bits at the leaves. Like our PAC algorithm, our EQ algorithm could be generalized to learn other classes with seeds.

Intuition and a warm-up. We first describe the intuition behind the algorithm and analyze a special case. In what follows let $q = 3\sqrt{n \ln s}$; we say that a seed is "well-sized" if it has size at most q. Recall that each seed T of a partial function f covers a non-empty subset of the positive examples of f. We define the *monomial associated with seed T of f* to be the conjunction of the literals l that are satisfied by all examples in that subset.

Let ϕ be a DNF of size s that represents the target function f. As a warm-up, let us first consider the simple case where each positive example of ϕ is covered by some well-sized seed T of f.[1] Note that the seed lemma does not guarantee that f has this property in general. Below we describe an equivalence query algorithm for this case. (This algorithm is equivalent to the online mistake-bound algorithm of Valiant (1999) for learning "projective functions" when the algorithm is applied to projections of at most q variables, taking projections to be monomials.)

We can divide the problem of learning ϕ into two tasks: determining which conjunctions T of at most q literals are seeds of f, and for each of those, learning the monomial T' associated with seed T of f. Having performed those two tasks, we can output the hypothesis that is the disjunction of all such terms T', since this disjunction is equivalent to ϕ.

The algorithm works by keeping a set of *candidate seeds*. Initially, this set consists of all conjunctions of literals of size at most q. The algorithm executes the following simple loop. Let S denote the set of counterexamples obtained so far (initially empty), and let f^S be the partial function defined by S (as in our description of the PAC algorithm). For each candidate seed T, the algorithm finds the subset S' of examples in S that satisfy T. It runs the standard PAC-learning algorithm for monomials on the sample S', which outputs a monomial T'. If S' has at least one positive example, then T' is the conjunction of the literals satisfied in all positive examples of S'; otherwise T' is the conjunction of all $2n$ literals (which is identically 0).[2]

The algorithm then checks whether T' is satisfied by any negative example in S. If so, then T is neither a seed of f^S nor a seed of f, and so it is removed from the set of candidate seeds. (If T' is the conjunction of all $2n$ literals, then T is not a seed of f^S, but it may be a seed of f, so we keep it as a candidate seed.) The algorithm then asks an equivalence query with the hypothesis h that consists of the disjunction of all T' associated with candidate seeds T of f^S. If it receives a counterexample, the algorithm runs the loop again.

The proof that this procedure eventually asks an equivalence query h equivalent to ϕ is based on the following observations. At each stage in the algorithm, the set of candidate seeds is always a superset of the set of actual seeds of f. Because of this, each hypothesis h is consistent with the current set S of counterexamples from which it was built. Thus each new counterexample must lead to a new hypothesis h. More particularly, each new positive counterexample must cause at least one literal to be removed from the constructed monomial T' for candidate seed T (which may or may not be a seed). Each new negative counterexample must disqualify at least one candidate seed that is not really a seed. For each actual seed T, the constructed monomial T' always contains a superset of the literals that it should contain (beginning with all $2n$ literals). Thus the algorithm makes progress with each counterexample, eventually ending up with only valid seeds T, correct monomials T', and a correct hypothesis h. This concludes our analysis of the "warm-up" case.

The general case. The algorithm for the general case is more complicated. Detailed pseudocode for the algorithm is given in Appendix C; in the rest of this section we provide a verbal description of the algorithm and analyze it.

1. The class of DNFs whose positive examples can all be covered by seeds of size at most k was studied by Sloan et al. (2008); such formulas are known as *k-projective DNF*. They gave a certificate result for this class, and noted that it implies a proper membership and equivalence query algorithm for learning the class (which is not computationally efficient).
2. We note that it is not actually necessary to run the PAC learning algorithm "from scratch" for each candidate T in each loop iteration; instead, whenever a new positive counterexample x is added to S' for some T, the algorithm can update T' by removing any literals not satisfied by x.

To motivate the algorithm, consider the following: Let Q be the set of all conjunctions of at most q literals. Suppose that we found all conjunctions of literals T of size at most q that were seeds of f from set Q. We could then create a new partial function f' from f, by changing the value of f to be undefined on all positive examples covered by the monomials T' associated with seeds T of f. Repeating this procedure until f' had no positive examples would yield a sequence of partial functions. Once a conjunction T was found to be a seed of one partial function in this sequence, it could never be a seed of a subsequent partial function in the sequence. Thus the length of the sequence is at most $|Q|$. Formally, we define a sequence of $|Q|$ partial functions (where the partial functions at the end of the sequence may have no positive examples) as follows: $f^{(1)} = f$, and for $1 < i \leq |Q|$, $f^{(i)}$ is defined recursively as follows: for all $x \in \{0,1\}^n$, $f^{(i)}(x) = *$ if x is a positive example of f and is covered by a well-sized seed T of $f^{(i-1)}$, and $f^{(i)}(x) = f^{(i-1)}(x)$ otherwise. (The simple case we analyzed above is the case where only $f^{(1)}$ has positive examples.)

At a high level, our algorithm works by keeping a set of candidate seeds for each $f^{(i)}$ in the sequence. For each candidate seed T of $f^{(i)}$, our algorithm also computes a monomial T'. The goal of the algorithm is to get to the point where, for each $f^{(i)}$, the set of candidate seeds consists precisely of all well-sized seeds T of $f^{(i)}$, and the computed T' for each such T is the monomial associated with seed T of $f^{(i)}$. At this point every positive example of f is covered by one of these T', and the hypothesis h that is the disjunction of the T' for all $f^{(i)}$ is equivalent to ϕ.

In more detail, for $1 \leq i \leq |Q|$ the set of candidate seeds for each $f^{(i)}$ is initialized to consist of all sets of at most q literals. The algorithm always maintains the invariant that the set of candidate seeds for each $f^{(i)}$ contains all actual seeds of $f^{(i)}$. The algorithm works by executing the following loop. Let S denote the set of counterexamples seen so far (initially empty). For each candidate seed T of $f^{(i)}$, let S_i denote the subset containing all negative examples in S, and all positive examples in S that do not satisfy any candidate seed of an $f^{(j)}$ where $j < i$. Note that an example x is a positive example of $f^{(i)}$ iff it does not satisfy a well-sized seed of $f^{(j)}$, for any $j < i$. Since the set of candidate seeds for each $f^{(i)}$ includes all well-sized seeds of $f^{(i)}$, let S_i denote the subset containing all positive examples in S that do not satisfy any candidate seed of an $f^{(j)}$ where $j < i$, and all negative examples in S. Thus S_i is a sample of $f^{(i)}$.

For each i, and for each candidate seed T of $f^{(i)}$, the algorithm computes the subset S' of examples in S_i that satisfy T. It then runs the standard PAC-learning algorithm for monomials on the sample S', which outputs a monomial T'. Next, it checks whether T' is satisfied by any negative example in S_i. If not, it eliminates T from the set of candidate seeds of $f^{(i)}$ (because T cannot be a seed of $f^{(i)}$).

The algorithm then sets hypothesis h to be the disjunction of the T' computed for candidate seeds T for all $f^{(i)}$. It asks an equivalence query with hypothesis h. If it receives a counterexample, it runs the loop again.

Correctness and running time of the algorithm. To see that this algorithm eventually constructs a correct hypothesis, observe first that in each loop iteration, the constructed hypothesis h is consistent with the counterexample set S. If a counterexample is received, it must either lead to a removal of a candidate seed of some $f^{(i)}$, or it must cause the removal of at least one literal in the T' constructed for a candidate seed T of some $f^{(i)}$. Further, if candidate seed T is actually a seed of $f^{(i)}$, the fact that S_i is a sample of $f^{(i)}$ guarantees that the removed literal does not belong in T'. Thus the algorithm makes progress toward its goal in each loop iteration.

Since each negative counterexample eliminates a candidate seed of some $f^{(i)}$, the number of negative counterexamples is $|Q| = 2^{O(\sqrt{n \log s} \log n)}$. Since each positive counterexample eliminates at least one literal from a term T' associated with a candidate seed of some $f^{(i)}$, the number of positive counterexamples is $n \times |Q| = 2^{O(\sqrt{n \log s} \log n)}$. Thus the algorithm will output a correct hypothesis in time $2^{O(\sqrt{n \log s} \log n)}$.

We have proved the following theorem.

Theorem 4 *The above algorithm properly EQ-learns DNF in time $2^{O(\sqrt{n \log s} \log n)}$.*

Our algorithm can be viewed as an MEQ algorithm that does not make membership queries, as can the (computationally inefficient) EQ algorithm presented in Section 6. Combining either one of these algorithms with the results of Hellerstein and Raghavan (2005) relating certificates and query complexity gives the following corollary. (We also give a direct proof, based on the seed lemma for DNF, in Appendix D.)

Corollary 5 *There exists a function $r(s,n) = 2^{O(\sqrt{n \log s} \log n)}$ such that for all $s, n > 0$, for all Boolean functions $f(x_1, \ldots, x_n)$, if dnf-size$(f) > r(s,n)$, then f has a certificate of size at most $r(s,n)$ certifying that $ds(f) > s$.*

6. A tradeoff between number of queries and size of queries

In this section we give a careful quantitative sharpening of the "approximate fingerprint" proof of Angluin (1990), which showed that poly(n)-term DNF cannot be properly EQ-learned with polynomial query complexity. We thereby prove a tradeoff between the number of queries and the size of queries that a proper EQ algorithm must use. Given any proper EQ algorithm A for learning DNF, we show that if A does not use hypotheses with many terms, then A must make many queries. Our result is the following (no effort has been made to optimize constants):

Theorem 6 *Let $17 \leq k \leq \sqrt{n/(2 \log n)}$. Let A be any EQ algorithm which learns the class of all poly(n)-size DNF formulas using queries which are DNF formulas with at most $2^{n/k}$ terms. Then A must make at least n^k queries in the worst case.*

Taking $k = \Theta(\sqrt{n/\log n})$ in Theorem 6, we see that any algorithm that learns $\mathrm{poly}(n)$-term DNF using $2^{\sqrt{n \log n}}$-term DNF hypotheses must make at least $2^{\Omega(\sqrt{n \log n})}$ queries.

We use the following lemma, which is a quantitative sharpening of Lemma 5 of (Angluin, 1990). The proof is in Appendix E.1.

Lemma 7 *Let f be any T-term DNF formula over n variables where $T \geq 1$. For any $r \geq 1$, either there is a positive assignment $y \in \{0,1\}^n$ (i.e. $f(y) = 1$) such that $|y| \leq r\sqrt{n}$, or there is a negative assignment $z \in \{0,1\}^n$ (i.e. $f(z) = 0$) such that $n > |z| > n - (\sqrt{n} \ln T)/r - 1$.*

Proof of Theorem 6: As in (Angluin, 1990) we define $M(n,t,s)$ to be the class of all monotone DNF formulas over $x_1, \ldots, x_n$ with exactly t distinct terms, each containing exactly s distinct variables. Let M denote $\binom{\binom{n}{s}}{t}$, the number of formulas in $M(n,t,s)$.

For the rest of the proof we fix $t = n^{17}$ and $s = 2k \log n$. We will show that for these settings of s and t the following holds: given any DNF formula f with at most $2^{n/k}$ terms, there is some assignment $a^f \in \{0,1\}^n$ such that at most M/n^k of the M DNFs in $M(n,t,s)$ agree with f on a^f. This implies that any EQ algorithm using hypotheses that are DNF formulas with at most $2^{n/k}$

terms must have query complexity at least n^k in the worst case. (By answering each equivalence query f with the counterexample a^f, an adversary can cause each such query to eliminate at most M/n^k of the M target functions in $M(n,s,t)$. Thus after n^k-1 queries at least $M/n^k > 1$ possible target functions in $M(n,t,s)$ must still be consistent with all queries and responses so far, so the algorithm cannot be done.)

Recall that $17 \le k \le \sqrt{n/(2\log n)}$. Let f be any DNF with at most $2^{n/k}$ terms. Applying Lemma 7 with $r = \sqrt{n}/2$, we get that either there is a positive assignment y for f with $|y| \le r\sqrt{n} = n/2$, or there is a negative assignment z with $n > |z| \ge n - (\sqrt{n}\ln(2^{n/k}))/r - 1 = n - \frac{(2\ln 2)n}{k} - 1 \ge n - \frac{3n}{k}$. Let ϕ be a DNF formula randomly and uniformly selected from $M(n,t,s)$. All probabilities below refer to this draw of ϕ from $M(n,t,s)$.

We first suppose that there is a positive assignment y for f with $|y| \le n/2$. In this case the probability (over the random choice of ϕ) that any fixed term of ϕ (an AND of s randomly chosen variables) is satisfied by y is exactly $\frac{\binom{y}{s}}{\binom{n}{s}} \le \frac{\binom{n/2}{s}}{\binom{n}{s}} \le \frac{1}{2^s}$. A union bound gives that $\Pr_\phi[\phi(y)=1] \le t/2^s$. Thus in this case, at most a $t/2^s$ fraction of formulas in $M(n,t,s)$ agree with f on y. Recalling that $t = n^{17}$, $s = 2k\log n$ and $k \ge 17$, we get that $t/2^s \le 1/n^k$ as was to be shown.

Next we suppose that there is a negative assignment z for f such that $n > |z| \ge n(1-\frac{3}{k})$. At this point we recall the following fact from (Angluin, 1990):

Fact 8 (Lemma 4 of (Angluin, 1990)) *Let ϕ be a DNF formula chosen uniformly at random from $M(n,t,s)$. Let z be an assignment which is such that $t \le \binom{n}{s} - \binom{|z|}{s}$.*[3] *Then $\Pr_\phi[\phi(z)=0] \le (1-((|z|-s)/n)^s)^t$.*

Since $t = n^{17}$, $|z| \le n-1$, and $s = O(\sqrt{n\log n})$, we indeed have that $t \le \binom{n}{s} - \binom{|z|}{s}$ as required by the above fact. We thus have

$$\Pr_\phi[\phi(z)=0] \quad \le \quad \left(1-\left(\frac{n(1-\frac{3}{k})-s}{n}\right)^s\right)^t = \left(1-\left(1-\frac{3}{k}-\frac{s}{n}\right)^s\right)^t.$$

Recalling that $k \le \sqrt{n/(2\log n)}$ we have that $s/n = 2k\log n/n \le 1/k$, and thus

$$\Pr_\phi[\phi(z)=0] \le \left(1-\left(1-\frac{4}{k}\right)^s\right)^t = \left(1-\left(1-\frac{4}{k}\right)^{2k\log n}\right)^{n^{17}}.$$

Using the simple bound $(1-\frac{1}{x})^x \ge 1/4$ for $x \ge 2$, we get that $\left(1-\frac{4}{k}\right)^{2k\log n} \ge 1/n^{16}$. Thus we have $\Pr_\phi[\phi(z)=0] \le \left(1-\frac{1}{n^{16}}\right)^{n^{17}} \le e^{-n} \ll \frac{1}{n^k}$ as was to be shown. ■

7. Achieving the tradeoff between number of queries and query size

In this section we prove a theorem showing that the tradeoff between number of queries and query size established in the previous section is essentially tight. Note that the algorithm A described in the proof of the theorem is not computationally efficient.

3. The statement of Lemma 4 of Angluin (1990) stipulates that $t \le n$ but it is easy to verify from the proof that $t \le \binom{n}{s} - \binom{|z|}{s}$ is all that is required.

Theorem 9 *Let $1 \leq k \leq \frac{3n}{\log n}$ and fix any constant $d > 0$. There is an algorithm A which learns the class of all n^d-term DNF formulas using at most $O(n^{k+d+1})$ DNF hypothesis equivalence queries, each of which is an $2^{O(n/k)}$-term DNF.*

Following Bshouty et al. (1996a), the idea of the proof is to have each equivalence query be designed so as to eliminate at least a δ fraction of the remaining concepts in the class. It is easy to see that $O(\log(|C|) \cdot \delta^{-1})$ such equivalence queries suffice to learn a concept class C of size $|C|$. Thus the main challenge is to show that there is always a DNF hypothesis having "not too many" terms which is guaranteed to eliminate many of the remaining concepts. This is done by taking a majority vote over randomly chosen DNF hypotheses in the class, and then showing that this majority vote of DNFs can itself be expressed as a DNF with "not too many" terms.

Proof of Theorem 9: At any point in the execution of the algorithm, let CON denote the set of all n^d-term DNF formulas that are consistent with all counterexamples that have been received thus far (so CON is the "version space" of n^d-term DNF formulas that could still be the target concept given what the algorithm has seen so far).

A simple counting argument gives that there are at most $3^{n^{d+1}}$ DNF formulas of length at most n^d. We describe an algorithm A which makes only equivalence queries which are DNF formulas with at most n^k terms and, with each equivalence query, multiplies the size of CON by a factor which is at most $\left(1 - \frac{1}{n^k}\right)$. After $O(n^{k+d+1})$ such queries the algorithm will have caused CON to be of size at most 1, which means that it has succeeded in exactly learning the target concept.

We first set the stage before describing the algorithm. Fix any point in the algorithm's execution and let $CON = \{f_1, \ldots, f_N\}$ be the set of all consistent n^d-term DNF as described above. Given an assignment $a \in \{0,1\}^n$ and a label $b \in \{0,1\}$, let $N_{a,b}$ denote the number of functions f_i in CON such that $f(a) = b$ (so for any a we have $N_{a,0} + N_{a,1} = N$), and let $N_{a,min}$ denote $\min\{N_{a,0}, N_{a,1}\}$.

Let Z denote the set of those assignments $a \in \{0,1\}^n$ such that $N_{a,min} < \frac{1}{n^k} \cdot N$, so an assignment is in Z if the overwhelming majority of functions in CON (at least a $1 - \frac{1}{n^k}$ fraction) all give the same output on the assignment. We use the following claim, whose proof is in Appendix E.2.

Claim 10 *There is a list of $t = \frac{3n}{k \log n}$ functions $f_{i_1}, \ldots, f_{i_t} \in CON$ which is such that the function $\mathsf{Maj}(f_{i_1}, \ldots, f_{i_t})$ agrees with $\mathsf{Maj}(f_1, \ldots, f_N)$ on all assignments $a \in Z$.*

By Claim 10 there must exist some function $h_{CON} = \mathsf{Maj}(f_{i_1}, \ldots, f_{i_t})$, where each f_{i_j} is an n^d-term DNF, which agrees with $\mathsf{Maj}(f_1, \ldots, f_N)$ on all assignments $a \in Z$. The function $\mathsf{Maj}(v_1, \ldots, v_t)$ over Boolean variables $v_1, \ldots, v_t$ can be represented as a monotone t-DNF with at most 2^t terms. If we substitute the n^d-term DNF f_{i_j} for variable v_j, the result is a depth-4 formula with an OR gate at the top of fanin at most 2^t, AND gates at the next level each of fanin at most t, OR gates at the third level each of fanin at most n^d, and AND gates at the bottom level. By distributing to "swap" the second and third levels of the formula from AND-of-OR to OR-of-AND and then collapsing the top two levels of adjacent OR gates and the bottom two levels of adjacent AND gates, we get that h_{CON} is expressible as a DNF with $2^t \cdot n^{dt} = 2^{O(n/k)}$ terms.

Now we can describe the algorithm A in a very simple way: at each point in its execution, when CON is the set of all n^d-term DNF consistent with all examples received so far as described above, the algorithm A uses the hypothesis h_{CON} described above as its equivalence query. To analyze the algorithm we consider two mutually exclusive possibilities for the counterexample a which is given in response to h_{CON}:

Case 1: $a \in Z$. In this case, since $h(a)$ agrees with the majority of the values $f_1(a), \ldots, f_N(a)$, such a counterexample causes the size of CON to be multiplied by at most $1/2$.

Case 2: $a \notin Z$. In this case we have $N_{a,0}, N_{a,1} \geq \frac{1}{n^k}$ so the counterexample a must cause the size of CON to be multiplied by at most $\left(1 - \frac{1}{n^k}\right)$. This proves Theorem 9. ■

8. Membership queries provably help for learning $\log n$-term DNF

The following is a sharpening of the arguments from Section 6 to apply to $\log(n)$-term DNF.

Theorem 11 *Let A be any algorithm which learns the class of all $\log n$-term DNF formulas using only equivalence queries which are DNF formulas with at most $n^{\log n}$ terms. Then A must make at least $n^{(\log n)/3}$ equivalence queries in the worst case.*

Sketch of Proof of Theorem 11: As in the proof of Theorem 6 we consider $M(n,t,s)$, the class of all monotone DNF over n variables with exactly t distinct terms each of length exactly s. For this proof we fix s and t both to be $\log n$. We will show that given any DNF formula with at most $n^{\log n}$ terms, there is an assignment such that at most a $1/n^{(\log n)/3}$ fraction of the DNFs in $M(n,t,s)$ agree with f on that assignment; this implies the theorem by the arguments of Theorem 6. Details are in Appendix E.3. ■

Acknowledgments

Lisa Hellerstein was supported by NSF Grant CCF-0917153. Devorah Kletenik was supported by the US Department of Education GAANN grant P200A090157. Linda Sellie was supported by NSF grant 0937060 to the CRA for the CIFellows Project. Rocco A. Servedio was supported by NSF grants CNS-0716245, CCF-0915929, and CCF-1115703.

References

M. Alekhnovich, M. Braverman, V. Feldman, A. Klivans, and T. Pitassi. Learnability and automatizability. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science*, pages 621–630, 2004.

Michael Alekhnovich, Mark Braverman, Vitaly Feldman, Adam Klivans, and Toniann Pitassi. The complexity of properly learning simple concept classes. *Journal of Computer & System Sciences*, 74(1):16–34, 2009.

Dana Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

Dana Angluin. Negative results for equivalence queries. *Machine Learning*, 5:121–150, 1990.

Dana Angluin. Computational Learning Theory: Survey and Selected Bibliography. In *Proceedings of the 24rd ACM Symposium on Theory of Computation*, pages 351–369, 1992.

Dana Angluin and Michael Kharitonov. When won't membership queries help? *Journal of Computer and System Sciences*, 50(2):336–355, 1995.

Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow - resolution made simple. *J. ACM*, 48 (2):149–169, 2001.

Avrim Blum and Steven Rudich. Fast learning of k-term DNF formulas with queries. *Journal of Computer and System Sciences*, 51(3):367–373, 1995.

Avrim Blum and Mona Singh. Learning functions of k terms. In *Proceedings of the 3rd Annual Workshop on Computational Learning Theory (COLT)*, pages 144–153, 1990.

Nader H. Bshouty. A Subexponential Exact Learning Algorithm for DNF Using Equivalence Queries. *Information Processing Letters*, 59(1):37–39, 1996.

Nader H. Bshouty. Simple learning algorithms using divide and conquer. *Computational Complexity*, 6:174–194, 1997.

Nader H. Bshouty, Richard Cleve, Richard Gavaldà, Sampath Kannan, and Christino Tamon. Oracles and queries that are sufficient for exact learning. *Journal of Computer and System Sciences*, 52(3):421–433, 1996a.

Nader H. Bshouty, Sally A. Goldman, Thomas R. Hancock, and Sleiman Matar. Asking questions to minimize errors. *J. Comput. Syst. Sci.*, 52(2):268–286, 1996b.

Andrzej Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, 82(3):231–246, 1989.

Oya Ekin, Peter L. Hammer, and Uri N. Peled. Horn functions and submodular boolean functions. *Theoretical Computer Science*, 175(2):257 – 270, 1997.

Lisa Hellerstein and Vijay Raghavan. Exact learning of DNF formulas using DNF hypotheses. *Journal of Computer & System Sciences*, 70(4):435–470, 2005.

Lisa Hellerstein, Krishnan Pillaipakkamnatt, Vijay Raghavan, and Dawn Wilkins. How many queries are needed to learn? *Journal of the ACM*, 43(5):840–862, 1996.

David P. Helmbold, Robert H. Sloan, and Manfred K. Warmuth. Learning nested differences of intersection-closed concept classes. *Machine Learning*, 5:165–196, 1990.

Adam Klivans and Rocco Servedio. Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. *Journal of Computer & System Sciences*, 68(2):303–318, 2004.

Hans-Ulrich Simon. Learning decision lists and trees with equivalence-queries. In *Proceedings of the Second European Conference on Computational Learning Theory*, pages 322–336, London, UK, 1995. Springer-Verlag. ISBN 3-540-59119-2. URL http://dl.acm.org/citation.cfm?id=646943.712223.

Robert H. Sloan, Balázs Szörényi, and György Turán. Projective dnf formulae and their revision. *Discrete Applied Mathematics*, 156(4):530–544, 2008.

Jun Tarui and Tatsuie Tsukiji. Learning DNF by approximating inclusion-exclusion formulae. In *Proceedings of the Fourteenth Conference on Computational Complexity*, pages 215–220, 1999.

L. Valiant. Projection learning. *Machine Learning*, 37(2):115–130, 1999.

Lelsie Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Appendix A. Learning models

In this appendix, we define the learning models used in this paper. We present the models here only as they apply to learning DNF formulas. See e.g. (Angluin, 1992) for additional information and more general definitions of the models.

In the PAC learning model of Valiant (1984), a DNF learning algorithm is given as input parameters ϵ and δ. It is also given access to an oracle $EX(c, \mathcal{D})$, for a target DNF formula c defined on X_n and a probability distribution $\mathcal{D}$ over $\{0,1\}^n$. On request, the oracle produces a *labeled example* $(x, c(x))$, where x is randomly drawn from distribution $\mathcal{D}$. An algorithm A *PAC-learns* DNF if for any DNF formula c on X_n, any distribution $\mathcal{D}$ on $\{0,1\}^n$, and any $0 < \epsilon, \delta < 1$, the following holds: Given ϵ and δ, and access to oracle $EX(c, \mathcal{D})$, with probability at least $1 - \delta$, A outputs a hypothesis h such that $\Pr_{x \in \mathcal{D}}[h(x) \neq c(x)] \leq \epsilon$. Algorithm A is a *proper* DNF-learning algorithm if h is a DNF formula.

In the EQ model of Angluin (1988), a DNF learning algorithm is given access to an oracle that answers *equivalence queries* for a target DNF formula c defined on X_n. An equivalence query asks "Is h equivalent to target c?", where h is a hypothesis. If h represents the same function as c, the answer is "yes," otherwise, the answer is a *counterexample* $x \in \{0,1\}^n$ such that $h(x) \neq c(x)$. If $c(x) = 1$, x is a *positive counterexample*, otherwise it is a *negative counterexample*. Algorithm A *EQ-learns* DNF if, for $n > 0$ and any DNF formula c defined on X_n, the following holds: if A is given access to an oracle answering equivalence queries for c, then A outputs a hypothesis h representing exactly the same function as c. Algorithm A EQ-learns DNF *properly* if all hypotheses used are DNF formulas.

A PAC or EQ learning algorithm *learns k-term DNF* if it satisfies the relevant requirements above when the target is restricted to be a k-term DNF formula.

In the *membership query* variants of the PAC and EQ models, the learning algorithm can also ask *membership queries*. In such a query the learning algorithm provides an assignment $x \in \{0,1\}^n$ and receives the value $c(x)$ of the target concept on x.

A PAC algorithm for learning DNF is said to run in time $t = t(n, s, \epsilon, \delta)$ if it takes at most t time steps, and its output hypothesis can be evaluated on on any point in its domain in time t, when the target is over $\{0,1\}^n$ and has size s. The time complexity for EQ algorithms is defined analogously for $t = t(n, s)$.

The *query complexity* of an EQ learning algorithm is defined to be the sum of the sizes of all hypotheses used.

Appendix B. Pseudocode for PAC algorithm of Section 4

We present the pseudocode for our PAC algorithm below, in Algorithm 1. In the pseudocode, we use S^+ (analogously, S^-) to denote the set of positive (negative) examples in S.

Algorithm 1: PAC algorithm

```
$X = \{x_1, \ldots, x_n\}, \bar{X} = \{\bar{x}_1, \ldots, \bar{x}_n\}$
$Q = \{t \subset X \cup \bar{X} \mid |t| \leq 3\sqrt{n \ln s}\}$ { set of potential seeds}
$h = 0$
while $Q \neq \emptyset$ AND $S^+ \neq \emptyset$ do
  for all $t \in Q$ do
    T = $\bigwedge_{l \in t} l$
    if $T$ covers at least one $e \in S^+$ then {test $T$ to see if it is a seed of $f^S$}
      $S_T = \{e \mid e \in S^+$ AND $T$ covers $e$ $\}$
      $T' = \bigwedge_{l \in B} l$ where $B = \{l \in X \cup \bar{X} \mid x$ is satisified by all $e \in S_T\}$.
      if $\{e \mid e \in S^-$ AND $e$ satisfies $T'\} = \emptyset$ then
        $S^+ = S^+ \setminus S_T$
        $h = h \vee T'$
        Remove $t$ from $Q$
      end if
    end if
  end for
end while
if $S^+ \neq \emptyset$ then
  return fail
else
  return $h$
end if
```

Appendix C. Pseudocode for EQ algorithm of Section 5

We present the pseudocode for our EQ algorithm below, in Algorithm 2. In the pseudocode, the set H_j contains each candidate seed T for $f^{(j)}$, with its current associated monomial T'. The condition $T' \not\equiv 0$ means that T' does not contain a variable and its negation. As in the previous section, let Q be the set of all terms of size at most $3\sqrt{n \ln s}$ (all potential seeds).

Appendix D. Subexponential certificates for functions of more than subexponential DNF size

We present a direct proof of Corollary 5, based on the seed lemma for DNF.

Proof Let $s, n > 0$. Let $q(s,n) = 3\sqrt{n \log s}$. Let f be a function on n variables such that *dnf-size*$(f) > (2n)^{q(s,n)}$. We first claim that there exists a partial function f', created by removing a subset of the positive examples from f and setting them to be undefined, that does not have a seed of size at most $q(s,n)$. Suppose for contradiction that all such partial functions f' have such a seed. Let S be the sample consisting of all 2^n labeled examples $(x, f(x))$ of f. We can apply the seed covering method of Section 4 to produce a DNF consistent with f, using a seed of size at most $q(s,n)$ at every stage. Since no seed will be used more than once, the output DNF is bounded by the number of terms of size at most $q(s,n)$, which is less than $(2n)^{q(s,n)}$. This contradicts that *dnf-size*$(f) > (2n)^{q(s,n)}$. Thus the claim holds, and f' exists.

Algorithm 2: EQ Algorithm

```
Initialize $h = 0$. Ask an equivalence query with $h$. If answer is yes **return** $h$, else let $e$ be the counterexample received.
for all $1 \leq j \leq |Q|$, $H_j = \{(T, T') \mid T \in Q, T' = \bigwedge_{l \in X \cup \bar{X}} l\}$
**while** True **do**
  **if** $e$ does not satisfy $h$ **then** $\{e$ is a positive counterexample$\}$
    **for** $j = 1, \ldots, |Q|$ **do**
      **if** $e$ satisfies $T$ for some $(T, T') \in H_j$ **then**
        **for all** $T$ such that $(T, T') \in H_j$ and $e$ satisfies $T$ **do**
          remove from $T'$ all literals falsified by $e$
        **end for**
        **break** out of **for** $j = 1, \ldots, |Q|$ loop
      **end if**
    **end for**
  **else** $\{e$ is a negative counterexample$\}$
    **for** $j = 1, \ldots, |Q|$ **do**
      Remove from $H_j$ all $(T, T')$ such that $T'$ is satisfied by $e$
    **end for**
  **end if**
  $H^* = \{T' :$ for some $j$, $(T, T') \in H_j$ and $T' \not\equiv 0\ \}$
  $h = \bigvee_{T' \in H^*} T'$
  Ask an equivalence query with hypothesis $h$. If answer is yes, **return** $h$, else let $e$ be the counterexample received.
**end while**
```

Since f' does not have a seed of size at most $q(s,n)$, each term T of size at most $q(s,n)$ either does not cover any positive examples of f', or the projection f'_T is not consistent with a monomial. Every function (or partial function) that is not consistent with a monomial has a certificate of size 3 certifying that it has that property, consisting of two positive examples of the function, and a negative example that is *between* them (cf. Ekin et al. (1997)). For assignments $r, x, y \in \{0,1\}^n$, we say that r is between x and y if $\forall i$, $p_i = r_i$ or $q_i = r_i$. It follows that if f'_T is not consistent with a monomial, then f' has a certificate $c(T)$ of size 3 proving that fact, consisting of two positive examples of f' that satisfy T, and one negative example of f' satisfying T that is between them.

Let $\mathcal{T} = \{T|$ term T is such that $|T| \leq q(s,n)$ and f'_T is not consistent with a monomial$\}$. Let $A = \bigcup_{T \in \mathcal{T}} c(T)$. Clearly $|A| < 3(2n)^{q(s,n)}$. We claim that A is a certificate that $\mathit{dnf\text{-}size}(f) > s$. Suppose not. Then there exists a function g that is consistent with f on the assignments in A, such that $\mathit{dnf\text{-}size}(g) \leq s$. Consider the partial function h which is defined only on the assignments in A, and is consistent with g (and f) on those assignments. The partial function h does not have a seed of size at most $q(s,n)$, because for all terms T of size at most $q(s,n)$, either T does not cover a positive assignment of h, or A contains a certificate that h_T is not consistent with a monomial. Since $\mathit{dnf\text{-}size}(g) \leq s$, and every DNF that is consistent with g is also consistent with h, $\mathit{dnf\text{-}size}(h) \leq s$ also. Thus by the seed lemma for DNF, h has a seed of size at most $q(s,n)$. This contradiction concludes the proof. ■

Appendix E. Proofs

E.1. Proof of Lemma 7

Proof of Lemma 7: The proof uses the following claim, which is established by a simple greedy argument:

Claim 12 (Lemma 6 of Angluin (1990)) *Let ϕ be a DNF formula with $T \geq 1$ terms such that each term contains at least αn distinct unnegated variables, where $0 < \alpha < 1$. Then there is a nonempty*[4] *set V of at most $1 + \lfloor \log_b T \rfloor$ variables such that each term of ϕ contains a positive occurrence of some variable in V, where $b = 1/(1-\alpha)$.*

Let f be a T-term DNF formula. Since by assumption we have $T \geq 1$, there is at least one term in f and hence at least one positive assignment y for f. If $r \geq \sqrt{n}$ then clearly this positive assignment y has $|y| \leq r\sqrt{n}$, so the lemma holds for $r \geq \sqrt{n}$. Thus we may henceforth assume that $r < \sqrt{n}$.

Let $\alpha = \frac{r}{\sqrt{n}}$ (note that $0 < \alpha < 1$ as required by Claim 12). If there is some term of f with fewer than $\alpha n = r\sqrt{n}$ distinct unnegated variables, then we can obtain a positive assignment y for f with $|y| < r\sqrt{n}$ by setting exactly those variables to 1 which are unnegated in this term and setting all other variables to 0. So we may suppose that every term of f has at least αn distinct unnegated variables. Claim 12 now implies that there is a nonempty set V of at most

$$1 + \lfloor \log_{1/(1-r/\sqrt{n})} T \rfloor \leq 1 + \frac{\sqrt{n}}{r} \ln T$$

variables V such that each term of f contains a positive occurrence of some variable in V. The assignment z which sets all and only the variables in V to 0 is a negative assignment with $n > |z| \geq n - (\sqrt{n} \ln T)/r - 1$ (note that $n > |z|$ because V is nonempty), and Lemma 7 is proved. ■

E.2. Proof of Claim 10

Proof Let functions $f_{i_1}, \ldots, f_{i_t}$ be drawn independently and uniformly from CON. (Note that $t \geq 1$ by the bound $k \leq \frac{3n}{\log n}$.) We show that with nonzero probability the resulting list of functions has the claimed property.

Fix any $a \in Z$. The probability that $\mathsf{Maj}(f_{i_1}, \ldots, f_{i_t})$ disagrees with $\mathsf{Maj}(f_1, \ldots, f_N)$ on a is easily seen to be at most

$$\binom{t}{t/2} \left(\frac{1}{n^k}\right)^{t/2} < \frac{2^t}{n^{kt/2}}.$$

Recalling that $t = \frac{3n}{k \log n}$, this is less than $1/2^n$ for all $1 \leq k \leq n$. Since there are at most 2^n assignments a in Z, a union bound over all $a \in Z$ gives that with nonzero probability (over the random draw of $f_{i_1}, \ldots, f_{i_t}$) the function $\mathsf{Maj}(f_{i_1}, \ldots, f_{i_t})$ agrees with $\mathsf{Maj}(f_1, \ldots, f_N)$ on all assignments in Z as claimed. ■

4. We stress that V is nonempty because this will be useful for us later.

E.3. Proof of Theorem 11

Proof of Theorem 11: Let $M(n,t,s)$ be the class of all monotone DNF over n variables with exactly t distinct terms each of length exactly s. Fix s and t both to be $\log n$. We will show that given any DNF formula with at most $n^{\log n}$ terms, there is an assignment such that at most a $1/n^{(\log n)/3}$ fraction of the DNFs in $M(n,t,s)$ agree with f on that assignment; this implies the theorem by the arguments of Theorem 6.

Let f be any DNF formula with at most $T = n^{\log n}$ terms. Applying Lemma 7 to f with $r = 1$, we may conclude that either there is an assignment y with $|y| \leq \sqrt{n}$ and $f(y) = 1$, or there is an assignment z with $n > |z| \geq n - \sqrt{n}(\log n)^2$ and $f(z) = 0$.

Let ϕ be a DNF formula randomly and uniformly selected from $M(n,t,s)$. All probabilities below refer to this draw of ϕ from $M(n,t,s)$.

We first suppose that there is an assignment y with $f(y) = 1$ and $|y| \leq \sqrt{n}$. The probability that any fixed term of ϕ (an AND of s randomly chosen variables) is satisfied by y is exactly

$$\frac{\binom{|y|}{s}}{\binom{n}{s}} \leq \frac{\binom{\sqrt{n}}{s}}{\binom{n}{s}} < \left(\frac{1}{\sqrt{n}}\right)^s = \frac{1}{n^{(\log n)/2}}.$$

A union bound gives that $\Pr_\phi[\phi(y) = 1] \leq t \cdot \frac{1}{n^{(\log n)/2}} < \frac{1}{n^{(\log n)/3}}$. So in this case y is an assignment such that at most a $\frac{1}{n^{(\log n)/3}}$ fraction of formulas in $M(n,t,s)$ agree with ϕ on y.

Next we suppose that there is an assignment z with $f(z) = 0$ and and $n > |z| > n - \sqrt{n}(\log n)^2$. Since $s = t = \log n$ and and $|z| \leq n-1$, we have that $t \leq \binom{n}{s} - \binom{|z|}{s}$ as required by Fact 8. Applying Fact 8, we get that

$$\begin{aligned}
\Pr_\phi[\phi(z) = 0] &\leq \left(1 - \left(\frac{n - \sqrt{n}(\log n)^2 - \log n}{n}\right)^{\log n}\right)^{\log n} \\
&< \left(1 - \left(\frac{n - 2\sqrt{n}(\log n)^2}{n}\right)^{\log n}\right)^{\log n} \\
&= \left(1 - \left(1 - \frac{2(\log n)^2}{\sqrt{n}}\right)^{\log n}\right)^{\log n} \\
&\leq \left(1 - \left(1 - \frac{2(\log n)^3}{\sqrt{n}}\right)\right)^{\log n} \\
&= \left(\frac{2(\log n)^3}{\sqrt{n}}\right)^{\log n} < \left(\frac{1}{n^{1/3}}\right)^{\log n} = \frac{1}{n^{(\log n)/3}}.
\end{aligned}$$

So in this case z is an assignment such that at most a $1/n^{(\log n)/3}$ fraction of formulas in $M(n,t,s)$ agree with ϕ on z. This concludes the proof of Theorem 11. ∎