How Many Missing Answers can be Tolerated by Query Learners? *

Hans Ulrich Simon

Fakultät für Mathematik, Ruhr-Universität Bochum, D-44780 Bochum, Germany
simon@lmi.ruhr-uni-bochum.de

Abstract. We consider the model of exact learning using an equivalence oracle and an incomplete membership oracle. In this model, a random subset of the learner's membership queries is left unanswered. Our results are as follows. First, we analyze the obvious method for coping with missing answers: search exhaustively through all possible "answer patterns" associated with the unanswered queries. Thereafter, we present two specific concept classes that are efficiently learnable using an equivalence oracle and a (completely reliable) membership oracle, but are provably not polynomially learnable if the membership oracle becomes slightly incomplete. The first class will demonstrate that the aforementioned method of exhaustively searching through all possible answer patterns cannot be substantially improved in general (despite its apparent simplicity). The second class will demonstrate that the incomplete membership oracle can be rendered useless even if it leaves only a fraction $1/\mathrm{poly}(n)$ of all queries unanswered. Finally, we present a learning algorithm for monotone DNF formulas that can cope with a relatively large fraction of missing answers (more than sixty percent), but is as efficient (in terms of run-time and number of queries) as the classical algorithm whose questions are always answered reliably.

1 Introduction

Certain classes of concepts, such as monotone DNF formulas and deterministic finite automata, have been shown to be polynomially learnable with equivalence and membership queries [1, 10, 2], but they are provably not polynomially learnable with equivalence queries alone [3]. Algorithms that rely upon a membership oracle are often of little pactical value because their questions cannot be answered reliably (even by human experts).[1] It seems natural to ask how many missing answers to membership queries can be tolerated by a (properly designed) learning algorithm. Angluin and Slonim [4] introduced the so-called "incomplete

* This work has been supported in part by the ESPRIT Working Group in Neural and Computational Learning II, NeuroCOLT2, No. 27150. The author was also supported by the Deutsche Forschungsgemeinschaft Grant SI 498/4-2.

[1] The analogous problem for the equivalence oracle can be circumvented by applying standard transformations in either the mistake bound or the pac-learning model [8, 2] (augmented by incomplete membership queries).

membership oracle", which behaves as follows. First, it is "persistent" in the sense that a question that is posed many times will always be answered (or left unanswered) in the same way. Second, it is "p-fallible" (for some fixed $0 \le p \le 1$) in the sense that a new question is left unanswered with probability p. Note that this model allows a smooth transition from learning with equivalence and membership queries (the case $p = 0$) to learning with equivalence queries alone (the case $p = 1$). A class being polynomially learnable in the former model but not in the latter must exhibit a "phase transition" at some "critical value" of p. For classes that are learnable in a robust sense, this phase transition will not happen before p approaches 1.

Example 1. Consider the class of monotone DNF formulas with n Boolean variables and m terms. There is a classical algorithm [2] that learns this class with n equivalence queries and at most mn membership queries. Angluin and Slonim [4] presented an algorithm that learns this class using an equivalence oracle and a p-fallible membership oracle. The expected number of queries and the run-time of their algorithm is polynomial in n and m for each constant $0 < p < 1$ (but superpolynomial in $1/(1-p)$). If $p = 1/2$, for instance, then the expected number of queries is $O(mn^2)$. Bshouty and Eiron [5] presented another learning algorithm for monotone DNF formulas whose expected total number of queries is $O(m^2n^2/(1-p)^2)$. Thus, their algorithm remains polynomial if p is a function of the form $1 - 1/\mathrm{poly}(n, m)$. Note that a $(1 - 1/\mathrm{poly}(n, m))$-fallible membership oracle may leave almost all queries unanswered (except for a polynomial fraction). Since membership oracles that answer only a superpolynomially small fraction of the queries are easily shown to be useless for polynomial learners, it follows that the class of monotone DNF formulas exhibits the largest possible robustness against incomplete membership oracles.[2]

In this paper, we explore to which extent query learners can be made robust against missing answers. After the formal definition of the learning model in Section 2, we present three main results:
In Section 3, we describe and analyze a general transformation of a given polynomial learner (expecting a completely reliable membership oracle) into a new polynomial learner that can cope with an expected number of $O(\log n)$ missing answers. The new learner performs an exhaustive search through all possible answer patterns associated with unanswered membership queries. The analysis has to deal with the issue of exhaustively searching through a space of random (and exponentially fast growing) size.
In Section 4, we present two specific concept classes that are efficiently learnable using an equivalence oracle and a (completely reliable) membership oracle, but are provably not polynomially learnable if the membership oracle becomes slightly incomplete. The first class will demonstrate that the aforementioned method of exhaustively searching through all possible answer patterns cannot be substantially improved in general (despite its apparent simplicity). The second

[2] Bshouty and Owshanko [6] have shown a similar result for the class of deterministic finite automata.

class will demonstrate that the incomplete membership oracle can be rendered useless even if it leaves only a fraction $1/\mathrm{poly}(n)$ of all queries unanswered. The derivation of the lower bounds on the query complexity of these classes must cope with some subtle issues because the learner can extract random bits from the answers of the incomplete membership oracle. For randomized learners, the standard adversary arguments do not readily apply.
In Section 5, we present a learning algorithm for monotone DNF formulas that can cope with a relatively large fraction of missing answers (more than sixty percent), but is as efficient as the classical algorithm whose questions are always answered reliably. In particular, the expected number of queries issued by this algorithm is $O(mn)$ for an arbitrary constant $p < (\sqrt{5}-1)/2 \approx 0.62$. Our algorithm results from a slight modification of the original algorithm of Angluin and Slonim [4]. In order to bound the expected number of queries by $O(mn)$, we have to apply a considerably refined analysis. It uses a method of "stripping-off" dependencies between random variables, which may be interesting in its own right.

2 The Learning Model

A *concept class* C over domain X is a set of functions of the form $f : X \to \{0, 1\}$. In the query-learning model, a learner has to identify an unknown function $f_* \in C$, called *target concept*, by means of queries that are honestly answered by an oracle. In this paper, we focus on the following types of queries:

Equivalence Queries (EQs) The learner passes a function $h \in C$ (functions outside C are not allowed!), called its *hypothesis*, to the equivalence oracle (EQ-oracle). The oracle answers either YES, signifying that $h \equiv f_*$, or returns a so-called *counterexample* $x \in X$ such that $h(x) \neq f_*(x)$.

Membership Queries (MQs) The learner passes an element $x \in X$ to the membership oracle (MQ-oracle). The oracle returns $f_*(x)$.

Incomplete Membership Queries (IMQs) The learner passes an element $x \in X$ to the incomplete membership oracle (IMQ-oracle). If an IMQ at query point x had already been issued before, then the oracle returns the same answer again. Otherwise, it flips a coin showing "heads" with some fixed probability p. If the coin shows "heads", the oracle returns "$*$" (= no answer). Otherwise, it returns $f_*(x)$. In order to indicate the probability parameter associated with the IMQ-oracle, we say that the IMQ-oracle is *p-fallible*.

Note that the EQ-oracle has, in general, many possibilities to answer honestly: it may return any counterexample x. Since we will expose the learning algorithms to a worst case analysis, we may assume that the EQ-oracle pursues an adversary strategy such as to slow down the learning progress as much as possible. In contrast to the EQ-oracle, the answer of the MQ-oracle (or IMQ-oracle) is determined by the query (and the outcome of the coin flip). Note that the IMQ-oracle flips its coins in an on-line fashion. Adversary strategies for the EQ-oracle can use the outcomes of past coin flips, but are ignorant to outcomes of future coin flips.

Formally, a *learner* for concept class C is an oracle algorithm, say A. In the EQ-model of learning, it is equipped with an EQ-oracle. An analogous remark is valid for the (EQ,MQ)-model and the (EQ,IMQ)-model of learning. An *adversary strategy* consists of rules that specify the target concept and the answers of the oracles, which may depend on the past interaction between the learner and the oracles (including the actual query). Learning then proceeds as an interactive process of the following kind:

1. First a target concept $f_* \in C$ (unknown to A) is fixed.
2. Then the program of A is executed. To each (allowed) query, the corresponding oracle returns an honest answer.

In query-learning models, it is usually assumed that the learner A is deterministic. In the (EQ,IMQ)-model however, even a deterministic program for A can extract random bits from the answers of the IMQ-oracle. For this reason, it is natural to allow randomized learners in this model. This has some subtle consequences. In a recent paper [11], it has been shown that the typical adversary arguments are no longer valid if A has access to random bits that cannot be predicted by the adversary.[3] Since the adversary will compete with a randomized learner, it will be convenient (for proof technical reasons) to consider randomized adversary strategies as well. The following definition takes these complications of the classical situation (of purely deterministic interactions) into account.

Definition 1. *We say that A (a potentially randomized oracle algorithm)* learns *C with an expected total number of q queries (of the allowed types) if, for any (potentially randomized) adversary strategy (concerning the choice of the target concept and the answers of the oracles), the following holds:*

1. *A issues at most q queries on the average (where the average is taken over all coin-flips of the IMQ-oracle, all internal coin-flips of the adversary strategy, and all internal coin-flips of A).*
2. *When A stops, target concept f_* is the only function from C that is consistent with all answers that were returned to A.*

Since asymptotic bounds will be an issue in this paper, we will consider parameterized concept classes and domains. (C_n) denotes a parameterized family of concept classes and concepts from C_n are functions from X_n to $\{0, 1\}$. A learning algorithm for (C_n) must be able to learn C_n for each $n \geq 1$. Its time bound and its (expected) total number of queries are then considered as functions depending on n (and sometimes depending also on an additional parameter representing the complexity of the target concept). A is called a *polynomial learner*

[3] For deterministic learners, it can be theoretically justified to evaluate the learner in a scenario where the adversary does *not* have to make the initial commitment to a target concept. Oracle answers are considered as honest as long they do not rule out any consistent explanation in C. For probabilistic learners however, the initial commitment is essential. See [11] for more information on this issue.

if its (expected) run-time is polynomially bounded and if it learns its concept class with an (expected) polynomially bounded number of queries. We will typically parameterize C such that $|X_n| = 2^{n(1+o(1))}$. For instance, for Boolean concept classes with $X_n = \{0,1\}^n$, n will denote the number of Boolean variables. This normalization condition rules out "dirty tricks" (such as realizing a small query bound in dependence on n by using a "crazy" parameterization). We will also apply the parameterization to the probability associated with the IMQ-oracle, i.e., we consider $p(n)$-fallible IMQ-oracles. Non-constant choices of p are particularly interesting when $p(n)$ approaches either zero or one. In the former case, the (EQ,IMQ)-model approaches the (EQ,MQ)-model. In the latter case, it approaches the EQ-model.

3 Coping with Few Missing Answers

In this section, we show that an algorithm which learns a concept class (C_n) with EQs and MQs can be made robust (to some extent) against a $p(n)$-fallible IMQ-oracle: as long as $p(n)$ does not exceed a critical threshold, there will be only a polynomially bounded loss in efficiency. In order to equip an algorithm with robustness to missing answers, we will apply the straightforward method of exhaustively searching through all possible "answer patterns". Note, however, that S missing answers lead to 2^S possible patterns. In other words, we will discuss an exhaustive search through a space of random (and exponentially fast growing) size.

Clearly, the expectation of 2^S is, in general, much different from $2^{E[S]}$. As for a binomially or normally distributed variable, we obtain the following result:

Lemma 1. *1. Let $S = S_{m,p}$ be the (binomially distributed) random variable that counts the number of successes in m independent Bernoulli trials, where each trial has probability p of success. Then $E\left[2^S\right] = (1+p)^m$.*

2. Let $N = N_{\mu,\sigma^2}$ be a normally distributed random variable with expectation μ and variance σ^2. Then $E\left[e^N\right] = e^{\mu+\frac{1}{2}\sigma^2}$.

Proof. 1. The proof is a simple application of the Binomial Theorem:

$$\begin{aligned} E\left[2^S\right] &= \sum_{k=0}^{m} \binom{m}{k} p^k (1-p)^{m-k} 2^k \\ &= (1-p)^m \cdot \sum_{k=0}^{m} \binom{m}{k} \left(\frac{2p}{1-p}\right)^k \\ &= (1-p)^m \cdot \left(1 + \frac{2p}{1-p}\right)^m = (1+p)^m \end{aligned}$$

2. Again, the proof is quite simple:

$$E\left[e^N\right] = \frac{1}{\sqrt{2\pi}\sigma} \cdot \int_{-\infty}^{+\infty} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \cdot e^x dx$$

$$= \frac{e^{\mu}}{\sqrt{2\pi}} \cdot \int_{-\infty}^{+\infty} e^{-\frac{1}{2}x^2} e^{\sigma x} dx$$
$$= \frac{e^{\mu+\frac{1}{2}\sigma^2}}{\sqrt{2\pi}} \cdot \int_{-\infty}^{+\infty} e^{-\frac{1}{2}(x^2-2\sigma x+\sigma^2)} dx = e^{\mu+\frac{1}{2}\sigma^2}$$

In the second equation, we applied the substitution theorem for integrals. In the last equation, we made use of $x^2 - 2\sigma x + \sigma^2 = (x-\sigma)^2$ and of the obvious equation

$$\frac{1}{\sqrt{2\pi}} \cdot \int_{-\infty}^{+\infty} e^{-\frac{1}{2}(x-\sigma)^2} dx = 1 \ .$$

•

We briefly note that a similar computation generalizes Lemma 1 for each $a > 0$ as follows:

$$E\left[a^S\right] = (1 + (a-1)p)^m$$
$$E\left[a^N\right] = a^{\mu + \frac{1}{2}\sigma^2 \ln(a)}$$

We now get back to the main subject of this section. We aim to transform a fault-intolerant learner (expecting completely reliable oracles) into a more fault-tolerant learner (being able to cope with an incomplete membership oracle to some extent). Let A be an algorithm with time bound $t(n)$ that learns (C_n) with $l(n)$ EQs and $m(n)$ MQs. Assume $t(n), l(n), m(n)$ are time-constructable.[4] We will design an algorithm A' that learns (C_n) with EQs and IMQs. The basic idea is quite simple:
A' proceeds like A except that MQs of A are passed to an IMQ-oracle; if an IMQ is left unanswered then both possible answers are returned (which increases the number of A-simulations that have to be pursued further).
As more and more IMQs are left unanswered, more and more A-simulations are run in parallel. In order to proceed with all these A-simulations in a well-organized fashion, A' uses an administrative tool: the so-called "simulation-tree" $\mathcal{T}$ (consisting only of a single root in the beginning). At any time, the leaves in $\mathcal{T}$ decompose into "active" and "inactive" leaves. Each "active" leaf v in $\mathcal{T}$ represents an active A-simulation $\mathcal{S}(v)$. Each "inactive" leaf v corresponds to an aborted simulation. Intuitively, A' will abort simulations that cannot be correct. Technically, A' checks whether an A-simulation respects the resource

[4] A function $f(n)$ is said to be *time-constructable* (some people say *countable*) if there is a deterministic Turing Machine (usually with one or two tapes, but we do not really need such a restriction) that, when started on input 1^n, outputs the binary representation $bin(f(n))$ of $f(n)$. This property can be used in the obvious way to make sure that a given resource bound $f(n)$ is respected. For instance, if we want to run a simulation for $f(n)$ steps, we could initialize a binary counter to $bin(f(n))$ and decrement it after each simulated step. When the counter reaches 0, the simulation is aborted. The family of time-constructable functions is quite rich. Assuming a function to be time-constructable is therefore not a severe restriction. See any standard book on complexity theory (like [7], for instance) for more details.

bounds $t(n), l(n), m(n)$. If not, it can be aborted without damage. Furthermore, A' checks a final hypothesis of an A-simulation for correctness. If it does not coincide with the target concept, then again the A-simulation can be aborted without damage. In order to control the resource bounds, A' keeps track of the following quantities: the number t_v of simulated steps within A-simulation $\mathcal{S}(v)$, the number l_v of EQs that A has issued in simulation $\mathcal{S}(v)$, and the number m_v of MQs that A has issued in simulation $\mathcal{S}(v)$. A high level description of A' (based on the simplifying assumption that A never issues the same query twice) reads as follows:

Initialization Create a simulation-tree $\mathcal{T}$ consisting of a single "active" root-node r. Initialize the quantities t_r, l_r, m_r to value 0.

Main Loop Select an active leaf v of lowest depth in $\mathcal{T}$ and proceed with the A-simulation $\mathcal{S}(v)$ — on the way, the quantities t_v, l_v, m_v are properly updated — until one of the following conditions holds:

a) *Resource-Bound-Condition:* $t_v > t(n)$, $l_v > l(n)$, or $m_v > m(n)$.
Corresponding Action: Abort $\mathcal{S}(v)$ and declare v as "inactive".

b) *Termination-Condition:* A-simulation $\mathcal{S}(v)$ terminates with final hypothesis h_*.
Corresponding Action: Invest an EQ to check whether h_* coincides with the target concept f_*. If $h_* \equiv f_*$, then return h_* and stop, else abort $\mathcal{S}(v)$ and declare v as "inactive".

c) *Blind-Spot-Condition:* An IMQ, say at query point x, is left unanswered.
Corresponding Action: Grow $\mathcal{T}$ by creating two "active" children of v: a left child v_0 and a right child v_1. In A-simulation $\mathcal{S}(v_0)$, classification label 0 is returned to the IMQ on x. Symmetrically, classification label 1 is returned in A-simulation $\mathcal{S}(v_1)$. Initialize the quantities $t_{v_0}, m_{v_0}, l_{v_0}, t_{v_1}, m_{v_1}, l_{v_1}$ properly.

We move on to the analysis of A'. Note that $\mathcal{T}$ contains exactly one designated path, starting from the root, that represents the "correct" A-simulation whose IMQs are always answered correctly. Thus, if the target concept is not accidentally identified within a wrong A-simulation, this path must finally lead to a leaf that satisfies the termination-condition with a correct final hypothesis. Note that the length of this path is bounded by the number S of unanswered IMQs within the correct A-simulation. Since, in the beginning of the main loop, A' always selects an "active" leaf v of lowest depth, the depth of $\mathcal{T}$ is also bounded by S, and the total number of leaves in $\mathcal{T}$ (= number of A-simulations that are run in parallel) is bounded by 2^S. Note that S is binomially distributed. These observations are employed in the proof of the following

Theorem 1. *Let A be an algorithm with time bound $t(n)$ that learns (C_n) with $l(n)$ EQs and $m(n)$ MQs. Assume $t(n), l(n), m(n)$ are time-constructable. Let A' be the corresponding robust algorithm (as described above). Assume the IMQ-oracle is $p(n)$-fallible. Then A' is an algorithm with expected-time bound $O((1 + p(n))^{m(n)} \cdot t(n))$ that learns (C_n) with an expected number of at most $(1 + p(n))^{m(n)} \cdot (l(n) + 1)$ EQs and with an expected number of at most $(1 +$*

$p(n))^{m(n)} \cdot m(n)$ IMQs. If $p(n) \leq k \cdot \log(n)/m(n)$ for some constant k, then the "blow-up" factor $(1+p(n))^{m(n)}$ is upper-bounded by the polynomial $n^{k\log(e)} \approx n^{1.44k}$.

Proof. We may assume without loss of generality that A issues exactly $m(n)$ MQs. Let $S = S_{m(n),p(n)}$ be the binomially distributed random variable that counts the total number of unanswered IMQs associated with the designated A-simulation whose blind spots are replaced by correct classification labels. Clearly, the random variable 2^S bounds the total number of A-simulations that are run in parallel by A' from above. Recall that A' aborts an A-simulation as soon as it runs for more than $t(n)$ steps, or as soon as it issues more than $l(n)$ EQs or more than $m(n)$ MQs. For this reason, the expected run-time of A' is bounded by $O(E[2^S] \cdot t(n))$. Similarly, the expected number of EQs issued by A' is bounded by $E[2^S] \cdot (l(n)+1)$, and the expected number of IMQs issued by A' is bounded by $E[2^S] \cdot m(n)$. Thus, the proof can be completed by an application of Lemma 1:

$$E\left[2^S\right] = (1+p(n))^{m(n)} < e^{p(n)\cdot m(n)}$$

Clearly, $p(n) \leq k \cdot \log(n)/m(n)$ implies that $e^{p(n)\cdot m(n)} \leq n^{k\log(e)}$. •

In the sequel, we will refer to A' as the *exhaustive search simulation* of A.

4 High Inherent Vulnerability to Missing Answers

In this section, we demonstrate the existence of concept classes that are efficiently learnable with EQs and MQs, but are provably not polynomially learnable if the membership oracle becomes slightly incomplete. In other words, each learning strategy is doomed to fail (in polynomial time) even if only few answers are missing. The first class that we employ for this purpose is called ADDRESSING (first considered by Maass and Turan [9]). The second class, that exhibits an even higher vulnerability to missing answers, will be called PARITY-ADDRESSING (and is actually a variant of the basic class ADDRESSING).

4.1 The Class ADDRESSING

We start with a formal description of the class ADDRESSING. For each fixed n, the domain $X_n = A_n \cup D_n$ of ADDRESSING consists of n "address points" $A_n = \{a_1, \ldots, a_n\}$ and 2^n "data points" $D_n = \{d_\alpha : \alpha \in \{0,1\}^n\}$. With each $\alpha \in \{0,1\}^n$, we associate the function f_α that maps a_i to α_i, $i = 1, \ldots, n$, d_α to 1, and all remaining data points to 0. $\text{ADDRESSING}_n = \{f_\alpha : \alpha \in \{0,1\}^n\}$.[5]

It has been observed by Maass and Turan [9] that (ADDRESSING_n) can be learned with n MQs, but at least $2^n - 1$ queries are needed if one learns with equivalence queries alone. The following result demonstrates that the two models

[5] The name ADDRESSING is used because α is a binary address of length n that uniquely determines 1-out-of-2^n data points.

are still separated by a superpolynomial gap if the EQ-oracle is augmented by an $\omega(\log(n)/n)$-fallible IMQ-oracle. If the IMQ-oracle is p-fallible for some constant $p > 0$, the gap is still exponential:

Lemma 2. *The expected total number of queries needed to learn the concept class (ADDRESSING$_n$) with EQs and $p(n)$-fallible IMQs is at least* $(1+p(n))^n/2$.

Proof. The expected number of queries needed for learning ADDRESSING$_n$ cannot increase if we change the learning model in favour of the learner as follows:

- In a first phase, the learner issues n IMQs on all the address points for free. Afterwards (phase 2), we start counting the number of queries that are still needed until the target concept is exactly identified.

Consider the following adversary strategy against any learner:

- Pick $\alpha \in \{0,1\}^n$ uniformly at random and commit yourself to target concept f_α.
- Upon query EQ(f_β), $\beta \neq \alpha$, return d_β as counterexample.

Let $S = S_{n,p(n)}$ be the binomially distributed random variable that counts the number of unanswered IMQs in the first phase of the learner. Let Q_u be the random variable that counts the number of queries in the second phase conditioned to the event that $S = u$. Finally, Q denotes the unconditioned random variable that counts the number of queries in the second phase. We claim that $E[Q_u] \geq 2^u/2$. Let us first show how the proof can be completed using this claim and Lemma 1:

$$E[Q] = \sum_{u=0}^{n} \Pr[S=u] \cdot E[Q_u] \geq \frac{1}{2} \cdot \sum_{u=0}^{m} \Pr[S=u] \cdot 2^u = \frac{1}{2} \cdot E[2^S] = \frac{1}{2} \cdot (1+p(n))^n$$

We finally sketch the proof for $E[Q_u] \geq 2^u/2$. The main argument is as follows. Since u answers are missing after phase 1, there are still 2^u functions in ADDRESSING$_n$ that are possible target functions. From the perspective of the learner, all of them are equally likely. Furthermore, each query in phase 2 — except for query EQ(f_α) — will rule out at most one of these functions, and the remaining ones are still equally likely.[6] Thus, on the average, it takes (at least) $2^u/2$ queries until the learner accidentally finds the target concept (by issuing query EQ(f_α)).[7] •

Consider the exhaustive search simulation of the algorithm that learns the class (ADDRESSING$_n$) with n MQs. According to Theorem 1, its expected total

[6] Formally, one can argue that all functions that are still possible target functions have the same *a posteriori* probability in the sense of Bayesian decision theory.

[7] If the learner issues only (irredundant) equivalence queries in phase 2, then the bound $(1+p(n))^n)/2$ is actually tight for phase 2.

number of queries is $(1+p(n))^n \cdot n$. [8] A comparison to the lower bound in Lemma 2 shows that no learner can perform substantially better. In particular, exhaustive search simulation is a polynomial learning method for (ADDRESSING$_n$) if the expected number of missing answers, which is actually $p(n)n$, satisfies $p(n)n = O(\log(n))$, or equivalently, if $p(n) = O(\log(n)/n)$. The lower bound in Lemma 2 shows that no algorithm can polynomially learn (C_n) if $p(n)$ goes beyond this barrier.

4.2 The Class PARITY-ADDRESSING

We aim to construct a concept class that (loosely speaking) exhibits the "highest possible" vulnerability to missing answers. More specifically, given an arbitrary integer constant $k \geq 1$, we will design a concept class (C_n) over a domain (X_n) with the following properties:

Property 1 For $q(n) = n^{k+1}$, $|X_n| = (n + 2^n)q(n) = 2^{n(1+o(1))}$.
Property 2 (C_n) is efficiently learnable with $nq(n)$ MQs and $q(n)-1$ EQs (and this is query-optimal).
Property 3 The expected total number of queries of each algorithm that learns (C_n) with EQs and n^{-k}-fallible IMQs asymptotically equals $2^{n-1}q(n)$.

This establishes an exponential gap between the (EQ,MQ)- and the (EQ,IMQ)-model even if the IMQ-oracle almost always answers: only a polynomially small fraction of the answers is missing on the average.[9]

As for the technical construction of a class with these properties, we use a variant of ADDRESSING where each address bit is computed as a parity of many other bits. If one of these many bits is not returned by the IMQ-oracle, then the parity function cannot be evaluated and the address bit remains unknown. For this reason, the class will be called PARITY-ADDRESSING.

We start with the specification of the domain. X_n decomposes into a set of $nq(n)$ "address points", A_n, and a (disjoint) set of $2^n q(n)$ "data points", D_n, respectively. More precisely, $X_n = A_n \cup D_n$, where

$$A_n = \{a_{i,j} : i = 1, \ldots, n, j = 1, \ldots, q(n)\} \ ,$$
$$D_n = \{d_{\beta,j} : \beta \in \{0,1\}^n, j = 1, \ldots, q(n)\} \ .$$

We say that data point $d_{\beta,j}$ has *binary address* β. With each Boolean matrix $B \in \{0,1\}^{n \times q(n)}$, we associate the binary address $\alpha(B) \in \{0,1\}^n$ such that

$$\alpha_i(B) = B_{i,1} \oplus \cdots \oplus B_{i,q(n)} \ .$$

In other words, $\alpha(B)$ is obtained from B by "XORing" the bits in each row of B. For each $B \in \{0,1\}^{n \times q(n)}$ and each $j' \in \{1, \ldots, q(n)\}$, we define the function

[8] In fact, if the simulation never issues the same query twice, it can be shown that it learns (ADDRESSING$_n$) with $O\left(n + (1 + p(n))^n\right)$ queries on the average.

[9] Note that superpolynomially small fractions of missing answers will necessarily blur the distinction between the two models.

$f_{B,j'} : X_n \rightarrow \{0,1\}$ by setting

$$f_{B,j'}(a_{i,j}) = B_{i,j} \ ,$$
$$f_{B,j'}(d_{\beta,j}) = \begin{cases} 1 & \text{if } \beta = \alpha(B) \text{ and } j = j' \ , \\ 0 & \text{otherwise} \ . \end{cases}$$

This leads to the concept class

$$\text{PARITY-ADDRESSING}_n = \left\{ f_{B,j'} : B \in \{0,1\}^{n \times q(n)}, j' \in \{1, \ldots, q(n)\} \right\} \ .$$

We now argue that PARITY-ADDRESSING$_n$ (in the role of C_n) has the three properties mentioned above. Property 1 is obvious. The following lemma is concerned with Property 2.

Lemma 3. *(PARITY-ADDRESSING$_n$) can be learned efficiently with $nq(n)$ MQs and $q(n) - 1$ EQs.*

Proof. Let $f_{B,j'}$ denote the target concept. Invest $nq(n)$ MQs to learn matrix B. Given B, compute $\alpha(B)$. Invest $q(n) - 1$ EQs for an exhaustive search through all $q(n)$ data points with binary address $\alpha(B)$. •

As a matter of fact, the learning method described in the proof of Lemma 3 is optimal:

Lemma 4. *(PARITY-ADDRESSING$_n$) cannot be learned (even with unbounded computational resources) with EQs and MQs if the total number of queries is smaller than $nq(n) + q(n) - 1$.*

Proof. Let $f_{B,j'} \in$ PARITY-ADDRESSING$_n$ denote the unknown target concept.[10] The progress of a learning algorithm can be measured by keeping track of the set $D'_n \subseteq D_n$ containing all data points d such that $f_{B,j'}(d) = 1$ is still conceivable. The proof easily follows from the following (easy to prove) observations:

1. Each EQ or each MQ on a data point reduces the size of D'_n at most by 1 (assuming an appropriate adversary).
2. We may assume without loss of generality that, for each $i \in \{1, \ldots, n\}$, a learner either explores all labels $f_{B,j'}(a_{i,1}), \ldots, f_{B,j'}(a_{i,q(n)})$ or none of them. In the former case, the learner made the "investment" of $q(n)$ MQs, but may derive the address bit $\alpha_i(B) = f_{B,j'}(a_{i,1}) \oplus \cdots \oplus f_{B,j'}(a_{i,q(n)})$ afterwards. In the latter case, the address bit $\alpha_i(B)$ remains unknown (in the strong sense that the learner can get no advantage over random guessing).[11]

[10] Choosing B uniformly at random from $\{0,1\}^{n \times q(n)}$ and j' uniformly at random from $\{1, \ldots, q(n)\}$ is a good adversary strategy.

[11] Note that this address bit would remain unknown if the learner invested fewer than $q(n)$ MQs on query points $a_{i,1}, \ldots, a_{i,q(n)}$. For this reason, the learner had better follow the "all or nothing" principle.

3. If all MQs are issued in the beginning (before the first EQ is issued), then the knowledge of a new address bit will halve the size of D'_n. In general (assuming an appropriate adversary), the knowledge of a new address bit cannot shrink the size of D'_n by a factor exceeding 2.
4. In order to shrink the size of D'_n from $2^n q(n)$ (the initial value) to 1 (the value after the identification of the target concept), the best one can hope for is reducing the size from $2^n q(n)$ to n by means of n halvings ($nq(n)$ MQs on address points), and then decrementing the size from $q(n)$ to 1 ($q(n)-1$ additonal queries on data points). (Using fewer "halvings" or using a "decrementation" before the last "halving" will slow down the learning progress.)

We omit further details. •

The final lemma is concerned with Property 3.

Lemma 5. *Let $p(n) = 1-(1-n^{-k})^{n^{k+1}} \approx 1-e^{-n}$. The expected total number of queries of each algorithm that learns (PARITY-ADDRESSING$_n$) with EQs and n^{-k}-fallible IMQs is at least $(1+p(n))^n q(n)/2$.*

Proof. The proof is similar to the proof of Lemma 2. We will sketch the main ideas. Let $f_{B,j'}$ denote the target concept.[12] We may assume that the $nq(n)$ IMQs on all address points are given for free to the learner. Remember that $\alpha_i(B) = f_{B,j'}(a_{i,1}) \oplus \cdots \oplus f_{B,j'}(a_{i,q(n)})$. Thus, address bit $\alpha_i(B)$ remains unknown (in the strong sense that the learner can get no advantage over random guessing) if one of the IMQs on $a_{i,1}, \ldots, a_{i,q(n)}$ is left unanswered, which happens with probability $p(n) = 1-(1-n^{-k})^{n^{k+1}}$. Since an address bit remains unknown with probability $p(n)$, there will be $(1+p(n))^n$ guesses (on the average) for $\alpha(B)$ that are still conceivable. From the persepctive of the learner, all these guesses are equally likely. Thus, there are $(1+p(n))^n q(n)$ (equally likely) guesses (on the average) for the unique data point $d_{\alpha(B),j'}$ that is mapped to 1 by the target concept. We may now argue (as in the proof of Lemma 2) that $(1+p(n))^n q(n)/2$ queries are needed (on the average) to complete the learning task. •

5 Monotone DNF Learning Revisited

We assume some familiarity with the theory of Boolean formulas. Recall that a monotone DNF formula is a disjunction of monotone terms (term = Boolean monomial). The class of monotone DNF formulas with at most m terms and n Boolean variables is denoted as MDNF$_{m,n}$. We impose the following lattice structure on the Boolean cube $\{0,1\}^n$:

$$x \leq y :\Leftrightarrow \forall i \in \{1, \ldots, n\} : x_i \leq y_i \ .$$

[12] Like in the proof of Lemma 4, we may assume that an adversary has chosen B and j uniformly at random.

For each Boolean point x, let x^i denote the point obtained from x by flipping the i-th coordinate (and keeping the other coordinates fixed). A point $x \in \{0,1\}^n$ is called *minimal point* of $f \in \mathrm{MDNF}_{m,n}$ if x is a minimal point in the Boolean lattice such that $f(x) = 1$. If f is minimized, the minimal points of f are in one-to-one correspondence with the monotone terms of f. In what follows, we identify f with its set of minimal points. Conversely, each $h \subseteq \{0,1\}^n$ is identified with the function from $\mathrm{MDNF}_{m,n}$ that maps x to 1 iff there exists a point $p \in h$ such that $x \geq p$, i.e., x must be located above a point from h in the Boolean lattice. Here is the main result of this section:

Theorem 2. *For each constant $p < c_0 := (\sqrt{5}-1)/2 \approx 0.62$, MDNF-LEARNER (described below) learns ($MDNF_{m,n}$) with EQs and p-fallible IMQs such that the expected total number of queries is bounded by*[13]

$$\frac{2mn}{1-p-p^2} + \frac{(1-p)m}{1-p-p^2} = O(mn) \ .$$

The remainder of this section is devoted to the proof of Theorem 2. We start with a somewhat technical but useful lemma:

Lemma 6. *Let X be a random variable with positive integer values. P denotes another random variable with (possibly multidimensional) "values" in some finite set $\mathcal{P}$.*[14] *For each $j \geq 1$, let Y_j denote a random variable with non-negative real values that satisfies $E[Y_j|X < j] = 0$ and*

$$\forall j \geq 1, \forall p \in \mathcal{P} : E[Y_j|X \geq j, P = p] \leq B \ .$$

for some bound $B \geq 0$. Then: $E\left[\sum_{j\geq 1} Y_j\right] \leq B \cdot E[X]$.

Proof. The proof is obtained from the following straightforward computation:

$$\begin{aligned}
E\left[\sum_{j\geq 1} Y_j\right] &= \sum_{j\geq 1} E[Y_j] \\
&= \sum_{j\geq 1} E[Y_j|X \geq j] \cdot \Pr[X \geq j] \\
&= \sum_{j\geq 1}\sum_{p\in\mathcal{P}} E[Y_j|P = p, X \geq j] \cdot \Pr[P = p|X \geq j] \cdot \Pr[X \geq j] \\
&\leq B \cdot \sum_{j\geq 1} \Pr[X \geq j] \cdot \left(\sum_{p\in\mathcal{P}} \Pr[P = p|X \geq j]\right) \\
&= B \cdot \sum_{j\geq 1} \Pr[X \geq j] \\
&= B \cdot \sum_{j\geq 1} \Pr[X = j] \cdot j = B \cdot E[X] \ .
\end{aligned}$$

•

[13] c_0 satisfies $c_0 + c_0^2 = 1$. Thus, $1 - p - p^2$ is a strictly positive constant.

[14] The type of this set will be irrelevant.

In our application of Lemma 6, X will count how many times a probabilistic procedure must be called until it is (for the first time) “succesful”. Y_j will count the number of queries during the j-th call of the procedure (which is zero by default if there is no j-th call because $X < j$). Random variable P will be chosen such that $E[Y_j|X \geq j, P = p] = E[Y_j|P = p]$. Loosely speaking, P allows us to strip off the statistical dependencies between Y_j and X. This will enable us derive a suitable upper bound B on $E[Y_j|X \geq j, P = p]$ and to apply Lemma 6.

We move on and present the algorithm MDNF-LEARNER (a slight modification of the original algorithm of Angluin and Slonim [4]):

```
algorithm MDNF-LEARNER
begin
  h ← ∅ ; WP ← ∅;
  while EQ(h) returns
          a counterexample x
  do if h(x) = 0
        then add newpoints(x) to h
        else remove wrongpoints(x)
               from h and add it to WP
     fi
  od
end

procedure wrongpoints(x)
begin
  return {p ∈ h : p ≤ x}
end

procedure newpoints(x)
begin
  Q ← {i ∈ {1,...,n} : x_i = 1};
  B ← ∅ ; BS ← ∅;
  while Q ≠ ∅
  do select and remove
       an element i from Q;
     if x^i ∈ WP
       then b ← 0
       else b ← IMQ(x^i)
     fi;
     case of
       b = 0 : do nothing
       b = * : add i to B
                 and x^i to BS;
       b = 1 : x ← x^i; Q ← Q ∪ B;
                 B = ∅
     esac
  od
  return {x}∪BS
end
```

MDNF-LEARNER has two global variables: h and WP. Both of them are initialized as empty sets. During the run of MDNF-LEARNER, the minimal points of the target concept f_* will be added to h, but, unfortunately, some negative examples of f_* will temporarily be added to h as well. Procedure **newpoints** is used to extend h until it contains all minimal points of f_*. Procedure **wrongpoints** is used to detect the “wrong points”, i.e., the negative examples of f_* that had been included into h. They are removed from h and stored in WP. The fact that WP contains only negative examples of f_* can be used to save redundant IMQs. A more detailed description follows.

Whenever MDNF-LEARNER gets a positive counterexample, say x, to its current hypothesis h, it calls the procedure **newpoints**(x). Procedure **newpoints**, when called on a positive counterexample x, tries to find a minimal point of the target concept f_* that is located below x in the lattice. It can work towards such a minimal point by flipping x_i from 1 to 0 whenever $f_*(x^i) = 1$. Technically, the algorithm keeps track of a set Q (initially containing the 1-coordinates of x) and (initially empty) sets B and BS (BS = Blind Spots). The elements of Q are inspected one by one (and removed from Q when they are inspected). If $x^i \in$ WP, then $f_*(x^i) = 0$ and the procedure stores label 0 in local variable b. Otherwise, an IMQ at query point x^i is issued and the returned label (possibly "$*$" if the query is left unanswered) is stored in b. Depending on the label that is stored in b different actions take place. If $b = 0$, there is nothing to do. If $b = *$, then i is inserted into B and x^i is inserted as "blind spot" into BS. If $b = 1$, then the procedure replaces x by x^i (thereby moving one step down in the lattice) and performs the updates $Q \leftarrow Q \cup B$ and $B \leftarrow \emptyset$. These actions make sure that the following holds after each iteration of the main loop in procedure `newpoints`:

- A coordinate i belongs to Q if $x_i = 1$ and the procedure did not yet try to determine $f_*(x^i)$ during the call.[15]
- A coordinate i belongs to B if $x_i = 1$, and an IMQ on query point x^i was left unanswered during the call.
- BS contains all blind spots that are created during the call of the procedure.

When the walk down the lattice gets stuck (which happens when $Q = \emptyset$), the current point x and all "blind spots" in BS are returned as new points (and then included in the current hypothesis h by the main program). We say that a call of `newpoints` at a positive counterexample x is *succesful* if the set `newpoints`(x) contains a minimal point of f_*. Note that the inclusion of blind spots in `newpoints`(x) increases the success probability, but it may insert negative examples of f_* into h. This is corrected by removing points from h that are located below a negative counterexample. (Compare with procedure `wrongpoints`.) Our analysis will use the following central notion. A *configuration* is a mapping $K : \{0,1\}^n \rightarrow \{0,1,*,-\}$ such that $K(x) = 0 \Rightarrow f_*(x) = 0$ and $K(x) = 1 \Rightarrow f_*(x) = 1$. $\mathcal{K}$ denotes the set of all configurations. Intuitively, $K(x) = -$ signifies that the IMQ at query point x has not yet been issued. The other labels indicate that this IMQ has already been issued, and that the label $0, 1$, or $*$ (no answer), respectively, had been returned. MDNF-LEARNER is started in *initial configuration* K_0 that maps each Boolean point to $-$. Note that the current configuration K and the coin flips of the IMQ-oracle completely determine the labels that are returned by the oracle. When the label $b \in \{0,1,*\}$ is returned upon an IMQ at query point x, the current configuration K is updated by setting $K(x) = b$. We are now in the position to analyze MDNF-LEARNER. The proof of Theorem 2 is completed by the following observations:

1. Since all blind spots are added to h (and removed only if they belong to WP), no positive counterexample is ever located above a blind spot p unless $p \in$ WP. This motivates the following definitions. The pair (K, x) is called

[15] It may be the case that coordinate i was already inspected during the same call (but not in connection with the same point x).

legal if $K(p) \neq *$ for all $p \leq x$ such that $p \notin \mathrm{WP}$. A call $\texttt{newpoints}(x)$ in configuration K is called a (K, x)-call of `newpoints`. It follows that MDNF-LEARNER initiates (K, x)-calls only for legal pairs (K, x). This has an important consequence: since $\texttt{newpoints}(x)$ issues an IMQ on a query point $y \leq x$ only if $y \notin \mathrm{WP}$, we can be sure that $K(y) \neq *$. The probability of getting no answer (answer "$*$") is therefore always bounded by p.
2. Assume that a call $\texttt{newpoints}(x)$ gets stuck at some point y. Remember that y and all blind spots in BS belong to the set $\texttt{newpoints}(x)$ that is returned upon termination. Point y must be located above a minimal point of f_* (which has not yet been included in hypothesis h). Let z be a minimal point of f_* that is located below y in the Boolean lattice and comes closest to y. Let s be the number of levels that separates y from z. If $s = 0$, then $y = z$ and $\texttt{newpoints}(x)$ is clearly succesful. If $s = 1$, then the search for a new minimal point of f_* tried to descend from y to z. Since the search got stuck without descending, the incomplete membership query at z was left unanswered (answered "$*$"). Thus z was included into the set BS of blind spots. Again, $\texttt{newpoints}(x)$ happens to be succesful. Thus, $s \geq 2$ for each call that is not succesful. Let $s(y) \geq s$ be the number of coordinates i in y such that y^i is still a positive example of f_*. Note that the probability of getting stuck at y is $p^{s(y)}$.
3. The preceding observations lead to the following conclusion. For each legal pair (K, x), the (K, x)-call of `newpoints` is not succesful with a probability of at most

$$\sum_{s=2}^{s(x)} p^s < \sum_{s=2}^{\infty} p^s = \frac{p^2}{1-p} \ ,$$

where the term p^s accounts for the event that the search for a minimal point of f_* below x gets stuck at some point y such that $s(y) = s$. Thus, the probability of being succesful exceeds $q = 1 - p^2/(1-p) = (1-p-p^2)/(1-p)$. Since $p < c_0$ and $c_0 = (\sqrt{5}-1)/2$ satisfies $c_0^2 + c_0 = 1$, we may conclude that $q > 0$. In case of success, a new minimal point of f_* is added to h. Thus, the expected number of positive counterexamples (= the number of times procedure `newpoints` is called) is smaller than $m/q = (1-p)m/(1-p-p^2)$.
4. Let (K, x) be a legal pair. Consider a (K, x)-call of `newpoints`. Since an IMQ is never issued on a query point y such that $K(y) = *$, at least every $1/(1-p)$-th IMQ (on the average) decrements the size of $Q \cup B$ by 1. Thus, the expected number of IMQs per legal call of `newpoints` is bounded by $n/(1-p)$.
5. The legal pairs will play the role of $\mathcal{P}$ in Lemma 6. We denote the random variable whose values are legal pairs by P. The preceding observation can now be restated as follows. If $Y_{K,x}$ denotes the random variable that counts the number of IMQs during the execution of a (K, x)-call of `newpoints`, then $E[Y_{K,x}] \leq n/(1-p)$ for each legal pair (K, x).
6. We decompose the run of MDNF-LEARNER into m phases. Each phase ends after a succesful call of `newpoints` (when a new minimal point of the target concept f_* has been inserted into the hypothesis h). Consider an arbitrary

but fixed phase i. Let X be the random variable that counts the number of calls of `newpoints` in phase i. Clearly, $E[X] < 1/q = (1-p)/(1-p-p^2)$. Let $P_j = (K(j), x(j))$ be the random variable (with legal pairs as values) such that the j-th call of `newpoints` in phase i is a P_j-call. Let Y_j be the random variable that counts the number of IMQs during the execution of this call (with default $Y_j = 0$ if `newpoints` is called fewer than j times in phase i). Thus, Y_j conditioned to $X < j$ is always zero. Recall that the current configuration $K(j)$ and the coin flips of the IMQ-oracle completely determine the labels that are returned by the oracle during the call `newpoints`$(x(j))$. The information how configuration $K(j)$ was reached (and in particular the fact that the last $j-1$ calls of `newpoints` were not succesful) are redundant given the information $(K(j), x(j))$. Thus, Y_j conditioned to $X \geq j$ and $P_j = (K(j), x(j))$ has the same probability distribution as Y_j conditioned to $P_j = (K(j), x(j))$. Furthermore Y_j conditioned to $P_j = (K(j), x(j))$ is distributed like $Y_{K(j),x(j)}$. We conclude that

$$E[Y_j | X \geq j, P_j = (K(j), x(j))] = E[Y_j | P_j = (K(j), x(j))] \leq n/(1-p) \ .$$

Lemma 6 implies that the expected total number of IMQs in phase i is at most $E[X] \cdot n/(1-p) = n/(q(1-p)) = n/(1-p-p^2)$. Summing over all m phases, we get that the expected total number of IMQs is bounded by $mn/(1-p-p^2)$.

7. Finally note that the total number of negative counterexamples is upper-bounded by the total number of blind-spots, which, in turn, is upper-bounded by the total number of IMQs. •

6 Open Problems

It would be interesting to know more robust learners for specific classes, and to know more natural examples of classes with high inherent vulnerability. Furthermore, we would like to know whether the barrier $(\sqrt{5}-1)/2$ from Theorem 2 can be pushed closer to 1.

Acknowledgements Thanks to an anonymous referee who found (and fixed!) two flaws in an earlier version of the paper and made numerous valuable suggestions.

References

1. Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.
2. Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.
3. Dana Angluin. Negative results for equivalence queries. *Machine Learning*, 5:121–150, 1990.
4. Dana Angluin and Donna K. Slonim. Randomly fallible teachers: Learning monotone DNF with an incomplete membership oracle. *Machine Learning*, 14:7–26, 1994.

5. Nader H. Bshouty and Nadav Eiron. Learning monotone DNF from a teacher that almost does not answer membership queries. In *Proceedings of the 14th Annual Workshop on Computational Learning Theory*, pages 546–557. Springer Verlag, 2001.
6. Nader H. Bshouty and Avi Owshanko. Learning regular sets with an incomplete membership oracle. In *Proceedings of the 14th Annual Workshop on Computational Learning Theory*, pages 574–588. Springer Verlag, 2001.
7. John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and their Relation to Automata*. Addison Wesley, 1969.
8. Nick Littlestone. Learning quickly when irrelevant attributes abound: a new linear threshold algorithm. *Machine Learning*, 2(4):245–318, 1988.
9. Wolfgang Maass and György Turán. Lower bound methods and separation results for on-line learning models. *Machine Learning*, 9:107–145, 1992.
10. Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993.
11. Hans U. Simon. How many queries are needed to learn one bit of information? In *Proceedings of the 14th Annual Workshop on Computational Learning Theory*, pages 1–13. Springer Verlag, 2001.