Available online at www.sciencedirect.com
SCIENCE DIRECT®
Artificial Intelligence 156 (2004) 139–176
Artificial Intelligence
www.elsevier.com/locate/artint

Theory revision with queries: Horn, read-once, and parity formulas

Judy Goldsmith[a,1], Robert H. Sloan[b,*,2], Balázs Szörényi[c], György Turán[c,d,3]

[a] *Computer Science Department, University of Kentucky, Lexington, KY 40506-0046, USA*
[b] *Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607-7053, USA*
[c] *Hungarian Academy of Sciences and University of Szeged, Research Group on Artificial Intelligence, Aradi vértanúk tere 1, H-6720 Szeged, Hungary*
[d] *Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, Chicago, IL 60607-7045, USA*

Received 6 March 2003; received in revised form 11 November 2003; accepted 15 January 2004

Abstract

A theory, in this context, is a Boolean formula; it is used to classify instances, or truth assignments. Theories can model real-world phenomena, and can do so more or less correctly. The theory revision, or concept revision, problem is to correct a given, roughly correct concept. This problem is considered here in the model of learning with equivalence and membership queries. A revision algorithm is considered efficient if the number of queries it makes is polynomial in the revision distance between the initial theory and the target theory, and polylogarithmic in the number of variables and the size of the initial theory. The revision distance is the minimal number of syntactic revision operations, such as the deletion or addition of literals, needed to obtain the target theory from the initial theory. Efficient revision algorithms are given for Horn formulas and read-once formulas, where revision operators are restricted to deletions of variables or clauses, and for parity formulas, where revision operators include both deletions and additions of variables. We also show that the query complexity of the read-once revision algorithm is near-optimal.
© 2004 Published by Elsevier B.V.

Keywords: Theory revision; Knowledge revision; Horn formulas; Query learning; Computational learning theory; Boolean function learning

[*] Corresponding author.
E-mail addresses: goldsmit@cs.uky.edu (J. Goldsmith), sloan@uic.edu (R.H. Sloan), szorenyi@rgai.hu (B. Szörényi), gyt@uic.edu (G. Turán).
[1] Partially supported by NSF grant CCR-0100040.
[2] Partially supported by NSF grants CCR-9800070 and CCR-0100336.
[3] Partially supported by NSF grants CCR-0100336 and CCR-9800070.

0004-3702/$ – see front matter © 2004 Published by Elsevier B.V.
doi:10.1016/j.artint.2004.01.002

1. Introduction

Sometimes our model isn't quite right. As computer scientists, we build models of real-world phenomena, based on limited data or on the opinions of sometimes-fallible experts. We verify or begin to use the models and discover that they are not quite correct. Rather than beginning the model-building phase again, we would prefer to quickly and simply revise the current model, and continue our project. If the initial model is nearly correct, this *should* be more efficient.

The revision of an initial theory, represented by a formula, consists of applying *syntactic revision operators*, such as the deletion or the addition of a literal. For instance, the CUP theory,[1] presented in Fig. 1, might be revised to become more accurate by deleting the literal white. The *revision distance* of the target theory from the initial theory is defined to be the minimal number of revision operations from a specified fixed set needed to produce a theory equivalent to the target, starting from the initial theory. As in our previous work [30] we consider two sets of revision operators: deletions-only operators, which allow the deletion of literals and of clauses and/or terms, and general operators, which also allow the addition of literals. Others have also implicitly or explicitly considered both of those models [32,40].

If the target theory is close to the initial theory, then an efficient revision algorithm should find it quickly. Thus, revision distance is one of the relevant parameters for defining the efficiency of theory revision.

One way of formalizing the problem of theory revision as a concept learning problem is: learn the class of concepts that are within a given revision distance of the initial theory. A novel feature of this definition is that it associates a concept class with *each concept*, and thus, in a sense, assigns a learning complexity to every individual concept (more precisely, to every concept representation, and every revision distance bound). This may perhaps help formalize the intuitive, yet elusive, notion that in general, there are hard and easy target concepts in learning theory. For instance, intuitively, there are hard and easy DNFs, but it does not make sense to talk about the difficulty of learning a particular DNF. On the other hand, it does make sense to talk about the difficulty of revising a particular DNF. So theory revision gives a way to quantify the learning complexity of each DNF.

This article and its companion article [30] consider revision in *query-based* learning models, in particular, in the standard model of learning with *membership* and *equivalence* queries, denoted by MQ and EQ [5]. This is a very well-studied model (e.g., [2,4–8,11,13–15]), nearly as much so as PAC-learning. In an equivalence query, the learning algorithm proposes a *hypothesis*, that is, a theory h, and the answer depends on whether $h = c$,

$$\begin{aligned}\mathsf{CUP} \equiv\ & \mathsf{has\text{-}concavity} \wedge \mathsf{white} \wedge \mathsf{upward\text{-}pointing\text{-}concavity} \wedge \mathsf{has\text{-}bottom}\\ & \wedge \mathsf{flat\text{-}bottom} \wedge \mathsf{lightweight} \wedge (\mathsf{has\text{-}handle} \vee (\mathsf{width\text{-}small} \wedge \mathsf{styrofoam}))\end{aligned}$$

Fig. 1. Cup theory/concept, inspired by Winston et al. [57]. Note that there may be many additional variables that are not used in the current cup theory.

[1] Cups are for theory revision what elephants are for computational learning theory and perhaps for AI in general, and what penguins are for nonmonotonic reasoning: the canonical toy example.

where c is the target theory. If so, the answer is “correct”, and the learning algorithm has succeeded in its goal of exact identification of the target theory. Otherwise, the answer is a *counterexample*: any instance x such that $c(x) \neq h(x)$. In a membership query, the learning algorithm gives an instance x, and the answer is either 1 or 0, depending on $c(x)$.

The *query complexity* of a learning algorithm is the number of queries it asks. Note that the query complexity is a lower bound on the running time. For running time, we do *not* count the time required to answer the queries. From a formal, theoretical point of view, we assume that there are two oracles, one each to answer membership and equivalence queries. In practice, membership queries would need to be answered by a domain expert, and equivalence queries could either be answered by a domain expert, or by using the hypothesis and waiting for evidence of an error in classification.

It is typical in practical applications that one starts with an initial theory and a set of (counter)examples, for which the initial theory gives an incorrect classification. The goal then is to find a small modification of the initial theory that is consistent with the examples. (In fact, many theory revision methods, including the algorithms presented here, would work even if a large number of changes were needed, but in that case it might be more efficient to learn from scratch rather than revising.) In this setup, one can simulate an equivalence query by running through the examples. If we find a counterexample to the current hypothesis, then we continue the simulation of the algorithm. Otherwise, we terminate the learning process with the current hypothesis serving as our final revised theory. In this way, an efficient equivalence and membership query algorithm can be turned into an efficient practical revision algorithm.

Besides this motivation, there are other reasons, specific to theory revision, that justify the use of equivalence and membership queries. In practical applications, it is often the case that the goal of theory revision is to fix an initial theory that is provided by an expert. It is reasonable to hope that the expert is able to answer further queries about the classification of new instances. For instance, natural language applications make this possibility apparent, as here everybody can serve as an expert, answering queries about the correctness of sentences. This means that in all these cases learning algorithms may be assumed to use membership queries.

Another important reason to study the query model is that it turns out to be the “right” model for many important learning problems. That is, for several basic problems, such as learning finite automata and Horn formulas, there are nontrivial efficient learning algorithms in this model, while in weaker models one can prove superpolynomial lower bounds.

In this paper we study two very important tractable classes of formulas: conjunctions of Horn clauses and read-once formulas.

Horn sentences are the tractable heart of several branches of computer science. The satisfiability of Horn sentences can be decided in polynomial—indeed linear—time (e.g., [23]). There is a combinatorial characterization of functions that can be expressed by Horn sentences [21,34,44,54].

Horn sentences have many applications. For instance, Horn sentences occur as special cases in logic, logic programming, and databases. Real-world reasoning and causality can be described by Horn theories: If the world is like so, then these are the consequences, separately and jointly. Horn sentences model safe queries in relational database theory [43].

Given the plethora of Horn sentences out there, it is imperative that we be able to mend those that are broken. The work presented in this paper is a first step in that direction.

Similarly to Horn formulas, read-once formulas form a nontrivial class that is tractable from several different aspects, but slight extensions are already intractable. Boolean functions represented by read-once formulas have a combinatorial characterization [33,35,46], and certain read restrictions make CNF satisfiability easily decidable in polynomial time (see, e.g., [38]). It is interesting that the tractable cases for fault testing [39] and Horn theory revision [24,40] are also related to read-once formulas.

The main results that we present in this paper are revision algorithms for Horn and read-once formulas in the deletions-only model of revisions, and a revision algorithm for parity functions in the general model of revisions. Some lower bounds are also provided.

The ultimate goal of this work is to revise real expert-system style theories, such as full Horn theories, using the types of queries we have already argued are feasible with real, human experts: membership and equivalence queries. In the course of pursuit of this as yet elusive goal, we have achieved some partial results, both for more restricted classes of theories and for more constrained revisions. Our work on parity formulas and on read-once formulas adds to a body of theoretical work on learning such formulas with queries (e.g., [8]), and showcases techniques and lower-bound proofs that we hope will be helpful in later work. They are included here as much for their mathematical elegance as for their eventual applicability.

We also include algorithms for revision constrained to deletions. We note that there is a long history of studying this special case, presumably because of its greater tractability, in, and even before, the AI literature. What we call "deletions only" corresponds to the "stuck-at" faults usually studied in diagnosing faulty circuits in the 1960s and 1970s (e.g., [39]) and, for instance, to the case where Koppel et al. proved the convergence of their empirical system for theory revision in the 1990s [40].

We note that there are scenarios where deletions-only would be quite useful for the intended application. Say, for example, that the more mature expert-system builder has designed a Horn-formula-based theory and sent her apprentice out to populate the theory. The apprentice interviews experts and enthusiastically writes down almost everything that each expert says, ignoring the experts' self-corrections.

It turns out that the model is imperfect, although the experts are sound.

Thus, the expert-system builder is faced with the task of revising the theory. From a review of the apprentice's methodology, it is clear that the revisions require only deletions.

As in the full revision model, the expert-system builder has access to the experts and may ask them the same types of queries. She could use the same algorithm to revise her theory, but she realizes that there is a more efficient algorithm available for the deletions-only case. This is precisely the algorithm that we present in Section 5.

In the next section we will discuss previous work on theory revision, and especially computational learning theory approaches to theory revision. Then, in Section 3, we discuss just what is meant by either the learning of or the revision of "propositional Horn formulas". We formally define basic concepts in learning with queries and in Boolean formulas in Section 4. In Section 5, we give our revision algorithm for propositional Horn formulas. We present our revision algorithm for read-once formulas in Section 6, and for parity formulas in Section 7.

2. Previous work

There is an extensive discussion of related work on theory revision in both the computational learning theory literature and the actual AI systems literature in our companion paper [30]. In this section, we briefly mention a few important, but somewhat arbitrarily selected, articles. In addition to our companion paper, we refer the reader to Wrobel's overviews of theory revision [58,59] for more detail. In the next section, we will discuss in more depth some papers that have each given results on something that they called "theory revision of propositional Horn formulas", although different researchers have actually considered quite different problems under that name.

Mooney [45] initiated the study of theory revision in computational learning theory using an approach based on syntactic distances. Mooney proposed considering the PAC-learnability of the class of concepts having a bounded syntactic distance from a given concept representation, and gave an initial positive result for sample complexity, but left computational efficiency as an open problem.

Numerous software systems have been built for theory revision. A few representative examples are EITHER [47], KBANN [52], PTR [40], and STALKER [18]. Many systems, such as STALKER, are designed for what Carbonara and Sleeman [18] have called the *tweaking* assumption: that the initial theory is fairly close to the correct theory. This would presumably be the case, for instance, when a deployed expert system is found to make some errors. On the other hand, KBANN can be viewed as solving essentially the usual general concept learning problem (using back-propagation for neural nets, and then sometimes translating back into propositional Horn sentences if possible), but starting the learning from some "in the ballpark" concept, instead of from some default "null concept". It is unclear whether KBANN's successes actually required initial theories that were "only a tweak" away from being correct.

Mooney implicitly assumes the tweaking case of theory revision. Here it is appropriate for the learning resources used (e.g., number of queries or sample size) to depend polynomially on the revision distance, but only subpolynomially on the size of the initial theory and the number of variables in the domain under consideration. For instance, we might want a dependence that is $\mathrm{O}(\log(\text{initial theory size} + n))$, where n is the number of variables in the domain. The reason this is desirable is that the tweaking assumption should mean that the revision distance $e \ll \max(\text{initial theory size},\ n)$, and we wish to revise using significantly fewer resources than learning from scratch.

These considerations also suggest a relationship between theory revision and *attribute-efficient learning* (see, e.g., [12,15,19,22]). Attribute-efficient learning is concerned with learning a concept from scratch, while using resources that depend polynomially on the number of variables in the target (called the number of *relevant variables*) and only logarithmically or polylogarithmically on the total number of variables in the universe. Roughly speaking, attribute-efficient learning is the special case of theory revision where the initial theory is the default, empty concept.

Angluin et al. [7] give a query-learning algorithm for Horn sentences. Our revision algorithm given in Section 5 is modeled on their algorithm. The primary difference between learning and revising Horn formulas, or any formulas, is the more stringent query complexity bounds required for revision, as opposed to learning from scratch. For instance, Angluin et al.'s algorithm to learn a Horn formula of n variables must ask $\Omega(n)$ queries, whereas we are limited to $o(n)$ queries.

Read-once formulas are efficiently learnable using equivalence and membership queries [8]. While read-twice DNF formulas are still efficiently learnable [48], for read-thrice DNF formulas there are negative results [1]. The query complexity of the learning algorithm for read-once formulas is $O(n^3)$, where n is the number of variables, or, equivalently, the length of the formula. In contrast, our revision algorithm for read-once formulas uses $O(e \log n)$ queries, where e is the revision distance between the initial and target formulas.

There has been a limited amount of work on theory revision for predicate logic. Greiner gives some results on theory revision in predicate logic in a paper that is primarily about revising propositional Horn formulas, which we discuss in the next section [32]. In another paper, Greiner [31] gives results about revision operators that change the order of the rules in a logic program. These, together with some results of Argamon-Engelson and Koppel [10] and Wrobel [58], are among the very few theoretical results on theory revision for predicate logic.

3. The dilemma of Horns

In the literature, "learning propositional Horn sentences" in fact refers to four distinct definitions of learning problems. Although there has been some discussion of this issue [3, 20,25], we think that some more clarification is possible, both for its own sake, and because it will clarify our discussion of the related work, especially Greiner's related work [31,32].

1. "Monotone circuit model". Propositional Horn sentences where only some of the variables are observable, and the problem is to classify an instance given only the values of those observable variables. The classification of instances over those observable variables depends on whether the sentence and the setting of the observable variables together imply a special output variable that occurs only as the head of clauses. This meaning is used in the EITHER theory revision system [47], and also in Mooney's theoretical PAC analysis of theory revision [45]. This model is equivalent to the model that complexity theorists call monotone circuits.
2. "Assignments model". Propositional Horn sentences where the classification of the instance depends on whether the assignment to (all) the variables agrees with or contradicts the Horn sentence. This meaning is used by Angluin et al. [7] and in this article, in our algorithm in Section 5.
3. "Entailment". Propositional Horn sentences where the instances themselves are *Horn clauses*, and a clause's classification depends on whether it is *entailed* by the target Horn sentence. This meaning is used by Frazier and Pitt in their work on *learning by entailment* [25,26], and also by Greiner [32].

$$\begin{aligned}
\text{CUP} &\leftarrow \text{Upright} \wedge \text{Liftable} \wedge \text{Open} \wedge \text{Graspable} \wedge \mathsf{white}\\
\text{Upright} &\leftarrow \mathsf{has\text{-}bottom} \wedge \mathsf{upward\text{-}pointing\text{-}concavity}\\
\text{Liftable} &\leftarrow \mathsf{lightweight} \wedge \mathsf{has\text{-}handle}\\
\text{Liftable} &\leftarrow \mathsf{width\text{-}small} \wedge \mathsf{styrofoam}\\
\text{Open} &\leftarrow \mathsf{upward\text{-}pointing\text{-}concavity}\\
\text{Graspable} &\leftarrow \mathsf{width\text{-}small}\\
\text{Graspable} &\leftarrow \mathsf{has\text{-}handle}
\end{aligned}$$

Fig. 2. A monotone-circuit style Horn sentence for CUP. It does not define exactly the same set as the definition in Fig. 1.

4. "Atomic entailment". The same entailment setting as 3, but now only *atoms* can be instances. Greiner also considers this case.

Consider Definition 1. An example of a Definition 1 style Horn formula for CUP (the example in Fig. 1 is of a read-once formula) is given in Fig. 2. The classification variable is CUP, and the hidden variables are UPRIGHT, LIFTABLE, OPEN, and GRASPABLE. One can describe such a sentence by a monotone circuit, with the classification variable corresponding to the output gate, the observables to inputs, and the hidden variables to interior gates. In fact, any monotone circuit is equivalent to such a Horn sentence.

Monotone circuits are a fairly rich class, and one that has been well studied in complexity theory. Monotone circuits are not learnable from equivalence queries alone, because the smaller class of monotone Boolean formulas is not learnable from equivalence queries alone [37].[2] To the best of our knowledge, it is an open question whether monotone circuits are learnable from membership and equivalence queries together.

Definition 2 is the one that we use for our revision result in Section 5. The cup example in Fig. 2 follows Definition 2 if all the variables including OPEN, GRASPABLE, etc., are visible. In general, in the Assignments model, training data for learning (or revising) from examples show the assignments to *all* the variables.

In Definition 3, entailment, there are again no hidden variables, but what is being learned is different. We might ask, in the cup example of Fig. 2, whether either of the following two clauses is entailed.

$$\begin{aligned}
\text{CUP} &\leftarrow \text{Liftable} \wedge \mathsf{upward\text{-}pointing\text{-}concavity}\\
\text{Liftable} &\leftarrow \mathsf{lightweight}.
\end{aligned}$$

(Note that neither example is entailed by the Horn sentence in Fig. 2.)

The main point of Model 4, entailment of atoms, is to use it to get strong negative results. Positive learning results would not be so useful, because Horn sentences over n propositional variables are an unreasonably large set of theories if all one wants to know is which of the n variables are positive and which are negative.

[2] One could also show the hardness of learning monotone circuits from equivalence queries by applying a standard variable substitution trick [36] to the cryptography result of Goldreich et al. that among other things says that the class of all polynomial-size circuits is not learnable from equivalence queries [27].

Angluin [3] provides some discussion on the comparison among the last three cases, as does Frazier [25]. In particular, Frazier shows how to convert a query learning algorithm for the assignment model into one for the entailment model. However, Frazier's conversions in general involve a multiplicative blowup in query complexity of the number of variables in the domain (i.e., of n), so the conversions cannot automatically transfer theory revision results for the assignments model into the entailment model. Further comparisons of the different approaches are given by De Raedt [20].

Greiner [32] considers Models 3 and 4, entailment of clauses and of atoms. Loosely speaking, Greiner shows that the non-tweaking cases of theory revision in Models 3 and 4 are NP-complete in the absence of membership queries, in both the general and deletions-only model. More precisely, he considers the PAC model, and so is interested in the decision problem of, given a sample of Horn clauses with each labeled either "entailed" or "not entailed", deciding whether there is a Horn sentence within a stated revision distance d that would so classify the clauses. Greiner shows that even for the entailment of atoms model, for $d = \Omega(\sqrt{|\varphi|})$, where $|\varphi|$ is the size of the initial theory φ, the problem is NP-complete. It is also nonapproximable, in the sense that one cannot find a Horn sentence that, say, agrees with 90% of the classifications of the sample, given usual complexity theory assumptions [32].

Note that Greiner's hardness results do not contradict our results. First, we allow membership queries in addition to sampling/equivalence queries. Some classes that have exponential query/sample complexity when only sampling/equivalence queries are used have polynomial query complexity when both membership and equivalence queries are used. Read-once Boolean formulas are an example of such a class [8,37]. On the other hand, arbitrary Boolean formulas are difficult to learn even with both membership and equivalence queries [9]. Another distinction between Greiner's negative results and ours is that we are primarily interested in smaller values of the revision distance than $d = \Omega(\sqrt{|\varphi|})$.

4. Preliminaries

We use the standard model of membership and equivalence queries (with counterexamples), denoted by MQ and EQ [5]. In an equivalence query, the learning algorithm proposes a *hypothesis*, a concept h and the answer depends on whether $h = c$, where c is the target concept. If so, the answer is "correct", and the learning algorithm has succeeded in its goal of exact identification of the target concept. Otherwise, the answer is a *counterexample*, any instance x such that $c(x) \neq h(x)$. For read-once formulas and parity functions, our equivalence queries will be *proper*; that is, the hypothesis will always be a revision of the initial formula. For Horn sentences, our equivalence queries will be "almost proper", meaning that the hypothesis will always be a conjunction of Horn clauses, with each Horn clause being a revision of a Horn clause in the initial formula, but there may be more than one revision of a single initial theory Horn clause in the hypothesis.[3] In a membership query,

[3] Our lower bound for Horn sentence revision will therefore also permit almost proper equivalence queries.

the learning algorithm gives an instance x and the answer is either 1 or 0, depending on $c(x)$, that is, $\mathrm{MQ}(x) = c(x)$, where again c is the target concept.

We also use standard notions from propositional logic such as variable, term, monotone, etc. We will assume throughout that the everywhere true and everywhere false formulas have some representation in each class of formulas that we study in this paper. The all-0 vector will be denoted $\mathbf{0}$; the all-1 vector $\mathbf{1}$. We occasionally use the standard partial order on vectors with $x \leqslant y$ if every component of x is less than or equal to the corresponding component of y.

The symbol $\subset$ always denotes *proper* subset.

A *Horn clause* is a disjunction with at most one unnegated variable; we will usually think of it as an implication and call the clause's unnegated variable its *head*, and its negated variables its *body*. We write $\mathrm{body}(C)$ and $\mathrm{head}(C)$ for the body and head of C, respectively. A clause with no unnegated variables will be considered to have head $\mathbf{F}$, and will sometimes be written as $(\mathit{body} \to \mathbf{F})$. A *Horn sentence* is a conjunction of Horn clauses.

For monotone terms s and t we use $s \cap t$ for the term that is the product of those variables in both s and t. As an example, $x_1x_2 \cap x_1x_3 = x_1$. (Thus $s \cap t$ is different from $s \wedge t$, which is the product of variables occurring in either s or t.)

When convenient, we treat Horn clause bodies as either monotone terms or as vectors in $\{0,1\}^n$, and treat vectors sometimes as subsets of $[n]$. If for $x \in \{0,1\}^n$ and Horn clause C we have $\mathrm{body}(C) \subseteq x$, we say x *covers* C. Notice that x *falsifies* C if and only if x covers C and $\mathrm{head}(C) \notin x$. (By definition, $\mathbf{F} \notin x$.)

Our Horn sentence revision algorithm makes frequent use of the fact that if x and y both cover clause C, and at least one of x and y falsifies C, then $x \cap y$ falsifies C.

A Boolean formula φ is a *read-once formula*, sometimes also called a μ-formula or a Boolean tree, if every variable has at most one occurrence in φ, and the operations used are $\wedge$, $\vee$, and $\neg$. Such a formula can be represented as a binary tree where the internal nodes are labeled with $\wedge$, $\vee$, and $\neg$ and the leaves are labeled with *distinct* variables or the constants 0 or 1. (For technical reasons, we extend the standard notion, which does not allow for constants in the leaves.) The internal nodes correspond to the subformulas. We call a subformula of φ *constant* if it computes a constant function. A constant subformula is maximal if it is not the subformula of any constant subformula.

By the de Morgan rules, we may assume that negations are applied only to variables. As we consider read-once formulas only in the deletions-only model, and thus know the sign of each variable—we can replace the negated variables with new variables (keeping in mind that every truth assignment should be handled accordingly). Thus without loss of generality we can assume each variable is unnegated (i.e., we use only $\wedge$ and $\vee$ in our read-once formulas). A Boolean function is read once if it has an equivalent read-once formula.

A *substitution* is a partial function $\sigma : \{x_1, \ldots, x_n\} \hookrightarrow \{0,1\}$. Given a substitution σ, let $\varphi\sigma$ be the formula obtained by replacing each variable x_i of φ that is in the domain of σ by $\sigma(x_i)$. Substitutions σ_1 and σ_2 are *equivalent* (*with respect to* φ) if $\varphi\sigma_1$ and $\varphi\sigma_2$ compute the same function.

For the lower bound on revising read-once formulas we shall use a well known notion, the *Vapnik–Chervonenkis* dimension [55] for Boolean functions. Let $\mathcal{C}$ be a set of Boolean functions on some domain X. We say that $Y \subseteq X$ is *shattered* by $\mathcal{C}$ if for any $Z \subseteq Y$ there is a $c_Z \in \mathcal{C}$ such that

$$c_Z(x) = \begin{cases} 1 & \text{if } x \in Z, \\ 0 & \text{if } x \in Y \setminus Z. \end{cases}$$

Then VC-dim$(\mathcal{C}) := \max\{|Y|\colon\ Y \subseteq X$ and Y is shattered by $\mathcal{C}$ is the *VC-dimension* of $\mathcal{C}\}$.

4.1. Theory revision definitions

Let φ be a Boolean formula using the variables $x_1, \ldots, x_n$. The concept represented by φ is the set of satisfying truth assignments for φ. For instance, if $\varphi = (x_1 \wedge x_2) \vee (x_1 \wedge x_3)$, then that concept would be $\{110, 101, 111\}$.

With the exception of Section 7, our revision operator is *fixing an occurrence of a variable* in a formula to a constant (i.e., to either 0 or 1). For instance, if we fix x_2 in φ to 1, we obtain the revised formula $(x_1 \wedge 1) \vee (x_1 \wedge x_3)$, which can be simplified to $x_1 \vee (x_1 \wedge x_3)$, and is equivalent to x_1. If instead we fix the second occurrence of x_1 to 0, we obtain the revised formula $(x_1 \wedge x_2) \vee (0 \wedge x_3)$, which can be simplified to $x_1 \wedge x_2$. Because the effect of fixing a literal to a constant for DNF and CNF formulas is to delete that literal, a clause, or a term, we also refer to this fixing of an occurrence of a variable to a constant as a *deletion*. Note that for read-once formulas, this instead corresponds to the "stuck-at" faults of fault detection.

For read-once formulas, where there is only one occurrence of the variable(s) being fixed, we write a revision using *substitution* notation, $\sigma = (x_i \to c_i)$, where c_i is a constant. For example, applying the substitution $\sigma = (x_2 \to 1, x_3 \to 1)$ to the formula $\varphi_2 = (x_1 \wedge x_2) \vee \neg x_3$ gives the revised formula $\varphi_2\sigma = (x_1 \wedge 1) \vee \neg 1$, which simplifies to x_1.

In Section 7, we also allow the *addition of a variable* as a revision operator. Handling this operator is more difficult, and the consideration of parity formulas provides an example of a tractable class.

We denote by $\mathcal{R}_\varphi$ the set of formulas obtained from φ by fixing some occurrences of some variables to constants. The corresponding concept class is denoted by $\mathcal{C}_\varphi$.

The *revision distance* between a formula φ and some concept $c \in \mathcal{C}_\varphi$ is defined to be the minimum number of applications of a specified set of revision operators to φ needed to obtain a formula for c. Thus, for example, we showed earlier that the revision distance between $\varphi = (x_1 \wedge x_2) \vee (x_1 \wedge x_3)$ and the concept represented by x_1 is 1.

A *revision algorithm* for a formula φ has access to membership and equivalence oracles for an unknown target concept $c \in \mathcal{C}_\varphi$ and must return some representation in $\mathcal{R}_\varphi$ of the target concept. Our goal is to find revision algorithms whose query complexity is polynomial in the revision distance between φ and the target, but at most *polylogarithmic* in the size of φ and the size of the variable set. The total *running time* of all our algorithms is always polynomial in the size of φ, the revision distance, and the number of attributes (since of course instances must be read and written). We do not explicitly calculate exact asymptotic running times because they are typically not drastically worse than the query complexity (e.g., number of attributes times query complexity) and because we expect the query complexity, or more generally, training data, to be the constraining factor in practical applications.

In fact, our results provide something stronger than a revision algorithm. The algorithms we give in this paper all revise some class of concept classes. That is, our algorithms are meta-algorithms, as they take any formula φ from a specified class of formulas (e.g., read-once formulas) and then function as a revision algorithm for the concept class $\mathcal{C}_\varphi$. Notice that the choice of revision operator(s) plays a double role. First, it defines the concept class: all things reachable from the specified formula with the revision operator(s). Second, it determines the revision distance, and that gives us a performance metric.

5. Revising propositional Horn sentences

In this section, we give an algorithm for revising Horn sentences in the deletions-only model. Angluin et al. [5] gave an algorithm for learning Horn sentences with queries. Their algorithm has query complexity $O(nm^2)$, where n is the number of variables and m is the number of clauses. This complexity is unacceptable for the revision task when the revision distance e is much smaller than the number of variables n. We give an algorithm, REVISEHORN, displayed as Algorithm 1, that has query complexity $O(em^3 + m^4)$ (independent of n).

In the following subsection we give more details about the algorithm; then, in Section 5.2, we give a lengthy example of a run of the algorithm. The reader may find it helpful to switch back and forth between the two subsections. The analysis of the query complexity and proof of correctness is in Section 5.3. In Section 5.4, we provide a lower bound.

5.1. Overview of algorithm

The highest-level structure of Algorithm REVISEHORN is similar to the structure of Angluin et al.'s algorithm for learning Horn sentences [5] and also to our DNF revision algorithm [30] (after making appropriate changes for the duality between the CNF form of Horn sentences and DNF form). The presentation in this section is self-contained; we do not assume familiarity with either of those papers.

We start with the hypothesis being the empty conjunction (i.e., everything is classified as true) and repeatedly, in an outer loop (lines 2–22), make equivalence queries until the correct Horn sentence has been found.[4] Each negative counterexample is used, with the help of membership queries made in subroutine SHRINKEXAMPLE, to make the hypothesis more restrictive; each positive counterexample is used to make the hypothesis more general.

We observe the following fact about negative instances, which we make implicit use of throughout.

[4] It is somewhat surprising that we start with the empty conjunction rather than the initial theory, but we have been unable to find a revision algorithm with good query complexity that starts with the initial theory.

```
1:  $h$ = empty hypothesis (everywhere true)
2:  **while** $(x = \mathrm{EQ}(h)) \neq$ “Correct” **do**
3:    **if** $h(x) == 1$ **then** {$x$ is a negative counterexample}
4:      $x = \textsc{ShrinkExample}(x, \varphi, h)$
5:      **for** each clause-group $C \in h$ in order **do**
6:        **if** $\mathrm{body}(C) \cap x \subset \mathrm{body}(C)$ **and then** $\mathrm{MQ}(\mathrm{body}(C) \cap x) == 0$
          **then**
7:          $\mathrm{body}(C) = x \cap \mathrm{body}(C)$
8:          **if** $\mathrm{head}(C) \neq \mathbf{F}$ **then**
9:            Add to $\mathrm{head}(C)$ any variable just deleted from $\mathrm{body}(C)$ that
              is the head of some clause of $\varphi$
10:         **end if**
11:         **break** the for loop
12:       **end if**
13:     **end for**
14:     **if** $x$ wasn’t used to shrink any clause-group in $h$ **then**
15:       $h = h \wedge (x \rightarrow \mathbf{F})$
16:     **end if**
17:   **else** {$x$ is a positive counterexample}
18:     **for** each clause $C$ of $h$ such that $C(x) = 0$ **do**
19:       **if** $\mathrm{head}(C) \neq \mathbf{F}$ **then**
20:         Delete $C$ from $h$
21:       **else**
22:         Change $C$ to clause-group with heads every head of a clause in
            $\varphi$ that is in $x \setminus \mathrm{body}(C)$
23:       **end if**
24:     **end for**
25:   **end if**
26: **end while**
27: **return** $h$
```

Algorithm 1. REVISEHORN. Revises Horn sentence φ.

Proposition 1. *Every negative instance of a CNF formula falsifies some clause of that CNF formula.*

Each negative counterexample is first processed by a subroutine called SHRINKEXAMPLE, Algorithm 2, which we will discuss in detail shortly. In general, that subroutine may change certain 1’s to 0’s while still leaving the negative counterexample as a negative counterexample to the current hypothesis.

Following Angluin et al., we sometimes find it convenient to organize our hypothesis by distinct clause *bodies*. We call the collection of all clauses in one Horn sentence that have the same body a *clause-group*. (Angluin et al. called a clause-group a *meta-clause*.) We will use the notation $(\mathit{body} \rightarrow x_1, x_4, x_5)$ to denote the clause-group with body *body* and heads x_1, x_4, and x_5, which is shorthand for the conjunction of clauses $(\mathit{body} \rightarrow x_1) \wedge (\mathit{body} \rightarrow x_4) \wedge (\mathit{body} \rightarrow x_5)$.

Algorithm REVISEHORN attempts to use the negative counterexample x returned from SHRINKEXAMPLE to make deletions from the body of an existing hypothesis clause-group C. This can be done when first $\mathrm{body}(C) \cap x \subset \mathrm{body}(C)$, so that there are some deletions to $\mathrm{body}(C)$ to make. We also need that $\mathrm{body}(C) \cap x$ is still a negative instance.

```
repeat
  done = true
  for each clause $C_0 \in \varphi$ do
    if $x \overset{\bullet}{\cap} \mathrm{body}(C_0) \neq x$ and then $\mathrm{MQ}(x \overset{\bullet}{\cap} \mathrm{body}(C_0)) == 0$ then
      $x = x \overset{\bullet}{\cap} \mathrm{body}(C_0)$
      done = false
    end if
  end for
until done
return $x$
```

Algorithm 2. SHRINKEXAMPLE(x, φ, h).

If so, then we update body(C) to body$(C) \cap x$, and, if any of the variables we are deleting from body(C) are possible heads, then we also add those variables as heads of C. For instance, if we have negative counterexample $x = 11000011$ and the hypothesis has clause-group $(x_1x_2x_3x_4 \to x_7, x_8)$ then, if $\mathrm{MQ}(x \cap x_1x_2x_3x_4) = \mathrm{MQ}(11000000) = 0$ and if x_3 and x_4 are both heads of some initial theory clauses, then this hypothesis clause-group is updated to $(x_1x_2 \to x_3, x_4, x_7, x_8)$.

If there is no hypothesis clause-group body that can be edited in that way, then we make the hypothesis more restrictive by adding a new clause to it, specifically $(x \to \mathbf{F})$. Notice that the very first counterexample will always add a new hypothesis clause.

Positive counterexamples are always used to make the hypothesis more general. We must somehow edit every hypothesis clause that is falsified by a positive counterexample. If a positive counterexample falsifies any hypothesis clause that has a head other than **F**, then that clause is simply deleted. In practice, this will have the effect of deleting some but not all the heads of a clause-group with multiple heads. (That fact follows from several lemmas that we prove in Section 5.3.)

If instead a positive counterexample x falsifies a clause-group C with head **F**, then this means that x covers C. In this case, C has some head(s) added to it, making it more general. In fact, we add all possible heads. Specifically, at line 22, REVISEHORN adds as heads of the clause-group C all heads of clauses of φ that correspond to 1's in x and are not in body(C).

5.1.1. Shrinking negative examples

The point of Algorithm SHRINKEXAMPLE is to take a negative counterexample x to the current hypothesis, and to decrease the Hamming weight of x. Ideally, x should contain only 1's in the positions corresponding to the body of the initial theory clause C_0 from which the target clause C_* that x falsifies is derived. Then if we use x to introduce a new hypothesis clause, that new hypothesis clause will not have too many extraneous variables in it. If instead we use x to make deletions from a hypothesis clause-group body, then a smaller counterexample is helpful because it produces more deletions.

We make the following observation, which we will use to help explain Algorithm SHRINKEXAMPLE.

```
1: $answer = x \cap y$
2: repeat
3:    for each clause $C$ in $h$ do
4:       if $x$ both covers and satisfies $C$ and $answer$ falsifies $C$ then
5:          Change head$(C)$ to 1 in $answer$
6:       end if
7:    end for
8: until $answer$ is not changed
```

Algorithm 3. $x \overset{\bullet}{\cap} y$ with respect to Horn sentence h.

Proposition 2. *If target formula clause C_* is a revision of initial theory clause C_0, then* $\text{body}(C_*) \subseteq \text{body}(C_0)$.

Proof. This follows because the only revision operator that we allow is deletion. □

Now notice that if negative counterexample x falsifies target clause C_* that is a revision of some initial theory clause C_0, then $x \cap \text{body}(C_0)$ also falsifies C_* because by Proposition 2, $\text{body}(C_*) \subseteq \text{body}(C_0)$. Thus, we would like to say that for each clause C_0 of the initial theory, if $\text{MQ}(x \cap \text{body}(C_0)) = 0$, then set x to $x \cap \text{body}(C_0)$. However, there is one issue to which we must pay careful attention.

We need to make sure that we do not, in the process of intersecting x with initial theory clause bodies, change x from an example that the current hypothesis classifies as positive to one the current hypothesis classifies as negative. This is why we use the funny notation $x \overset{\bullet}{\cap} C_0$ instead of $x \cap C_0$ in lines 4 and 5 of SHRINKEXAMPLE, which we now explain.

Let h be a collection of Horn clauses. The $\overset{\bullet}{\cap}$ operation with respect to h (which will usually be understood to be the current constructed hypothesis) is formally defined to be the result of the pseudocode given as Algorithm 3.

The idea is that the result of $x \overset{\bullet}{\cap} y$ is the same as the result of $x \cap y$ *except* when there are one or more hypothesis clauses C such that $x \cap y$ covers $\text{body}(C)$ and x has a 1 in the position $\text{head}(C)$, in which case that 1 stays on regardless of y, for each such hypothesis clause.

Example. Imagine that the current hypothesis is $h = (x_1x_2 \to x_5) \wedge (x_1x_2 \to x_6) \wedge (x_5x_6 \to x_7)$. (Notice, by the way, that this hypothesis contains three clauses but only two clause-groups.) Now $1111111 \cap x_1x_2x_3 = 1110000$, but $111111 \overset{\bullet}{\cap} x_1x_2x_3 = 1110111$. The first loop of the $\overset{\bullet}{\cap}$ operation will set *answer* to 1110110, and the second to 1110111.

We make an easy observation about the $\overset{\bullet}{\cap}$ operation, and then next we prove that the $\overset{\bullet}{\cap}$ operation has the property we want in terms of making sure that its output satisfies the hypothesis.

Proposition 3. *For any x and y,*

$$x \cap y \subseteq x \overset{\bullet}{\cap} y \subseteq x.$$

Lemma 4. *If x satisfies hypothesis h, then for any y, the instance $x \overset{\bullet}{\cap} y$ with respect to h also satisfies h.*

Proof. There are two different ways an instance can satisfy a Horn clause: either by not covering the clause, or by covering the clause and having the clause's head set to 1. We know that x satisfies every clause c_h of h. If x does not cover c_h, then neither can $x \overset{\bullet}{\cap} y$, because $x \overset{\bullet}{\cap} y \subseteq x$.

If x does cover c_h, then x has head(c_h) set to 1 because x satisfies c_h. Now the procedure for $\overset{\bullet}{\cap}$ guarantees that if $x \overset{\bullet}{\cap} y$ covers c_h, then $x \overset{\bullet}{\cap} y$ will satisfy c_h by having head(c_h) set to 1. □

The other interesting thing about Algorithm SHRINKEXAMPLE is that it *repeatedly* loops through all the initial theory clauses, continuing to look for deletions from x until we make a full pass through all the initial theory clauses without changing x at all. We need this repeated looping to guarantee a property of the output of SHRINKEXAMPLE that is proved later in Lemma 8.

5.2. *An example run of* REVISEHORN

We now give an example run of REVISEHORN. Suppose the variable set is $\{x_1, x_2, x_3, x_4, x_5\}$, and the target formula φ and the target formula ψ are given by

$$\varphi \equiv (x_1x_2x_3 \to x_4) \wedge (x_2x_4 \to x_1) \wedge (x_2x_4 \to x_5),$$
$$\psi \equiv (x_1x_2x_3 \to x_4) \wedge (x_2x_4 \to x_1) \wedge (x_2 \to x_5).$$

Algorithm REVISEHORN always initializes the hypothesis h to the everywhere true empty conjunction. Say EQ$(h) = 11101$, a negative counterexample.

So now we call SHRINKEXAMPLE$(11101, \varphi, h)$. It first determines that $11101 \overset{\bullet}{\cap} x_1x_2x_3 = 11100 \neq 11101$, so it asks the query MQ(11100) and learns that 11100 is also a negative instance, so x is reset to be 11100. Next $11100 \overset{\bullet}{\cap} x_2x_4 = 01000 \neq 11100$, so the query MQ$(01000) = 0$ is made, and x is reset to 01000. Since the third initial formula clause has the same body as the second, $01000 \overset{\bullet}{\cap} x_2x_4 = 010000$. Now SHRINKEXAMPLE begins the second iteration of its main loop. This time $01000 \overset{\bullet}{\cap} x_1x_2x_3 = 01000$ and $01000 \overset{\bullet}{\cap} x_2x_4 = 01000$, so x is not altered, and the value 01000 is returned.

Accordingly, the hypothesis is updated by REVISEHORN to be

$$h = (x_2 \to \mathbf{F}).$$

The next main loop of REVISEHORN makes the equivalence query EQ(h), and this time say EQ$(h) = 11111$, a positive counterexample. Since the hypothesis clause has head $\mathbf{F}$, at line 22 REVISEHORN puts in all possible heads, updating the hypothesis to:

$$h = (x_2 \to x_1, x_4, x_5).$$

Now suppose that $\mathrm{EQ}(h) = 11001$. This positive counterexample causes the hypothesis clause $(x_2 \to x_4)$, which it falsifies, to be deleted. The hypothesis is updated to:

$$h = (x_2 \to x_1, x_5).$$

This time say $\mathrm{EQ}(h) = 11101$. Now in SHRINKEXAMPLE, $11101 \overset{\bullet}{\cap} x_1x_2x_3 = 11101$ (and *not* 11100 because 11101 would falsify the hypothesis clause $(x_2 \to x_5)$). Next $11101 \overset{\bullet}{\cap} x_2x_4 = 11001$, and so the membership query $\mathrm{MQ}(11001) = 1$ is made. Since that membership query returns 1, SHRINKEXAMPLE does not modify its input at all, and returns 11101, and the hypothesis now becomes

$$(x_2 \to x_1, x_5) \wedge (x_1x_2x_3x_5 \to \mathbf{F}).$$

Now say $\mathrm{EQ}(h) = 11111$, a positive counterexample. We change the heads of the second hypothesis clause-group so the hypothesis is now:

$$(x_2 \to x_1, x_5) \wedge (x_1x_2x_3x_5 \to x_4).$$

Now say $\mathrm{EQ}(h) = 01111$, another positive counterexample. REVISEHORN removes x_1 as a head of the first hypothesis clause-group, updating the hypothesis to

$$(x_2 \to x_5) \wedge (x_1x_2x_3x_5 \to x_4).$$

Say this time $\mathrm{EQ}(h) = 01111$. When SHRINKEXAMPLE is called, it first determines that $01111 \overset{\bullet}{\cap} x_1x_2x_3 = 01101$ and $\mathrm{MQ}(01101) = 1$, so that does not change x. Next, $01111 \overset{\bullet}{\cap} x_2x_4 = 01011$, and $\mathrm{MQ}(01011) = 0$, so x is changed to 01011. No further changes to x are made in SHRINKEXAMPLE, so 01011 is returned by SHRINKEXAMPLE. Now back in REVISEHORN, $x_2 \cap 01011 = x_2$, so editing the first hypothesis clause-group is not considered. Next $x_1x_2x_3x_5 \cap 01011 = x_2x_5$, so the membership query $\mathrm{MQ}(01001) = 1$ is tried, but since it returns 1, the second hypothesis clause-group is also not edited. Instead, a new clause-group is added, giving the hypothesis

$$(x_2 \to x_5) \wedge (x_1x_2x_3x_5 \to x_4) \wedge (x_2x_4x_5 \to \mathbf{F}).$$

Now say $\mathrm{EQ}(h) = 11011$. Then REVISEHORN will use this positive counterexample to change the third clause-group, and we will arrive at

$$(x_2 \to x_5) \wedge (x_1x_2x_3x_5 \to x_4) \wedge (x_2x_4x_5 \to x_1).$$

Finally, $\mathrm{EQ}(h) =$ “Correct”. Notice, by the way, that the final correct hypothesis does not have exactly the same form as we stated, but is equivalent to it via resolution.

5.3. Horn revision algorithm correctness

Once we have established that Algorithm REVISEHORN halts, its correctness follows from its form. We prove a bound on its query complexity using a series of lemmas.

Several of these lemmas involve proving that some property of the hypothesis is invariant. We point out here that there are only four places where the hypothesis is ever changed: one place where hypothesis clause-group bodies are created, one where they can be altered, and two places where the set of heads of a clause-group can be altered. One is using a positive counterexample to edit the hypothesis in lines 18–24 of REVISEHORN. The other is moving a clause-group body variable into the head of the clause-group at Line 9 of REVISEHORN.

We begin with an observation about the heads of the hypothesis clauses. We then prove several facts about SHRINKEXAMPLE, which is at the heart of making the query complexity independent of the number of variables n.

Proposition 5. *Every head of a hypothesis clause-group other than* **F** *is a head of some initial theory clause.*

We record in the following lemma the fact that x remains a negative counterexample to the current hypothesis after it is modified in SHRINKEXAMPLE.

Lemma 6. *If x is a negative instance satisfying Horn sentence h, then the instance returned by* SHRINKEXAMPLE(x, φ, h) *is also a negative instance satisfying h.*

Proof. As the algorithm proceeds, x is modified to be $x \overset{\bullet}{\cap} \text{body}(C_0)$ only immediately after a membership query guarantees that $x \overset{\bullet}{\cap} \text{body}(C_0)$ is a negative instance. Thus the returned instance will be negative.

Lemma 4 says that if x satisfies h, then so does $x \overset{\bullet}{\cap} y$ for any y. □

Next, before proceeding to bound the query complexity of the entire REVISEHORN algorithm, we bound the query complexity of the SHRINKEXAMPLE algorithm.

Lemma 7. *Algorithm* SHRINKEXAMPLE *makes at most* $\mathrm{O}(m^2)$ *membership queries, where m is the number of clauses in the initial theory φ.*

Proof. Each iteration of the outer **repeat until** loop makes at most one query for each clause in φ. To prove the lemma, we will prove that there are at most $2m + 1$ iterations of the outer **repeat until** loop.

Each time there is an iteration of the outer loop, x must be altered. The only way that x is ever altered is by changing 1's to 0's. For a given initial formula clause $C_0 \in \varphi$, once we set $x = x \overset{\bullet}{\cap} \text{body}(C_0)$, we know that we have set to 0 every position of x that is not either in body(C_0) or the head of some hypothesis clause.

Thus, x can be altered at most once for each head of a hypothesis clause, plus once for each initial theory clause. The heads of the hypothesis clauses are a subset of the heads of the initial theory clauses, so there are at most m of them, as there are m initial theory clauses. Thus x can be altered at most $2m$ times, so the outer loop can execute at most $2m + 1$ times, as desired. □

Now we show how the output of SHRINKEXAMPLE is connected to the notion of revision of the initial formula.

Lemma 8. *Let x be the output from* SHRINKEXAMPLE. *For every target clause C_* that x falsifies, for each initial theory clause C_0 such that C_* is a revision of C_0, any position that* *is a* 1 *in* x *either corresponds to a variable in* $\text{body}(C_0)$ *or corresponds to a head of some initial theory clause.*

Proof. Assume for contradiction that for some such C_0 and C_* that x contains a 1 in a position that is neither in $\text{body}(C_0)$ nor a head of an initial theory clause. Consider the final iteration of the outer **repeat until** loop of Algorithm SHRINKEXAMPLE. Note that x is unchanged in the final iteration of the algorithm. We will derive a contradiction by showing that this x would be changed.

By our assumption about x, we know that $x \overset{\bullet}{\cap} \text{body}(C_0) \neq x$. If we can show that $x \overset{\bullet}{\cap} \text{body}(C_0)$ is a negative instance of the target, we have our contradiction, because then this x would be modified at line 5 of SHRINKEXAMPLE, forcing another iteration of the outer **repeat until** loop.

Now x falsifies C_*, so x already has $\text{head}(C_*)$ set to 0. Therefore, $x \overset{\bullet}{\cap} \text{body}(C_0)$ also has $\text{head}(C_*)$ set to 0. If we now show that $x \overset{\bullet}{\cap} \text{body}(C_0)$ covers C_*, then we have shown that $x \overset{\bullet}{\cap} \text{body}(C_0)$ is a negative instance, and we are done. Because x falsifies C_*, we have that x covers C_*. Since C_* is a revision of C_0, we have $\text{body}(C_*) \subseteq \text{body}(C_0)$. By Proposition 3, $x \cap \text{body}(C_0) \subseteq x \overset{\bullet}{\cap} \text{body}(C_0)$, so $x \overset{\bullet}{\cap} \text{body}(C_0)$ covers C_*. □

Now we move on to show that every clause-group body falsifies at least one clause of the target Horn sentence, and that no target clause is falsified by more than one clause-group body. We first prove the first of these two facts, and then prove some lemmas we will need to prove the second.

Lemma 9. *Each clause-group body in the hypothesis always falsifies some clause of the target concept.*

Proof. The body of the clause-group is always a negative instance of the target. This is true when the clause-group is first added at line 15 of SHRINKEXAMPLE by Lemma 6, and this is maintained as an invariant because it is guaranteed by a membership query immediately before changing a clause-group body at line 7 of REVISEHORN. □

Lemma 10. *For every hypothesis clause-group* C *with head other than* **F**, *for every target clause* C_* *that* $\text{body}(C)$ *falsifies,* $\text{head}(C_*)$ *is always one of the heads of* C.

Proof. When created, every hypothesis clause-group has head **F**.

When we first change the clause-group C's head from **F**, we know by Lemma 9 that $\text{body}(C)$ falsifies at least one target clause. Also, we know from the existence of a counterexample that covers $\text{body}(C)$ and is classified by the target as positive that the target clauses that are falsified by $\text{body}(C)$ must have a head other than **F**. At this point we put in all possible heads. When we delete a variable from a clause-group body, if it is a possible head, we add it. We remove a head only when a positive counterexample guarantees that it must be removed. □

From this lemma we can show that no clause-group in the hypothesis is ever altogether deleted from the hypothesis, although it may be revised in various ways.

Corollary 11. *No hypothesis clause-group, once introduced, is ever deleted.*

Proof. The only way that this could potentially happen would be if at line 20 of REVISEHORN we removed the last clause (i.e., head) of a particular hypothesis clause-group C.

Consider hypothesis clause-group C. If head$(C) = \mathbf{F}$, then it has the one head $\mathbf{F}$, and there is no operation to remove it.

If C has head(s) other than $\mathbf{F}$, then by Lemma 9, body(C) falsifies some target clause C_*. Now, by Lemma 10, one of the heads of clause-group C is head(C_*). There cannot be any positive counterexample falsifying the hypothesis clause whose body is body(C) and whose head is head(C_*), so that head of clause-group C is never deleted. □

Lemma 12. *For a given hypothesis clause-group C, no variable is ever added as head more than once.*

Proof. Heads are initially added once when a clause-group head is first changed from $\mathbf{F}$ at line 22 of REVISEHORN. After that happens, the clause-group will never have head $\mathbf{F}$ again. Thereafter, heads are added when they are deleted from the body. Because there are no additions made to hypothesis bodies, these heads could not previously have been heads of that hypothesis clause; they were always in the body. Once deleted from the body, they are never restored. □

Lemma 13. *No two hypothesis clause-group bodies ever falsify the same target clause.*

Proof. We follow the proof in Angluin et al. [5] of an analogous statement about their algorithm for learning Horn sentences from scratch.

We first show that the following claim implies the lemma, and then prove the claim.

Claim. *Consider the clause-group bodies $b_1, b_2, \ldots, b_h$ of hypothesis h in the order added. For any j, if b_j falsifies target clause C_*, then no b_i with $i < j$ covers C_*.*

Assume that the claim is true, but nevertheless the bodies of hypothesis clause-groups C_k and C_ℓ both falsify the target clause C_*, and WLOG, $k < \ell$. This contradicts the claim, since body(C_k) falsifies C_*.

Now we prove that the claim is true by induction on the number of changes made to the hypothesis. It is certainly vacuously true of the initial empty hypothesis. We must show that this property remains invariant whenever we alter the hypothesis. Positive counterexamples do not change the set of clause-group bodies, so we need consider only negative counterexamples.

Consider first the case of modifying a clause-group body b_j at line 7 of Algorithm REVISEHORN by setting $b_j = x \cap b_j$, where $b_j = $ body(C_j). After this modification, b_j cannot cover more clauses of the target formula than before, so we need worry only about clause-groups b_i with $i < j$. Suppose for contradiction that b_j now falsifies some new target clause C_*, and b_i covers C_*, with $i < j$. It must be that before this change that b_j covered C_* and so x falsified C_*. Now we have that $b_i \cap x$ falsifies C_*, because b_i covers C_* and x falsifies C_*. Therefore x would have been used to edit b_i in the **for** loop at lines 5–13 of REVISEHORN, as long as $b_i \cap x \subset b_i$. What happens if $b_i \cap x = b_i$? Since $b_i \cap x = b_i$ falsifies C_*, we have that b_i falsifies C_*. By Lemma 10, b_i's clause-group either has head **F** or has head(C_*) among its heads. Therefore x does not satisfy b_i's clause-group, because x covers b_i and x falsifies C_*. Lemma 6 says that x must satisfy (every clause of) the hypothesis.

Next, consider the case of adding a new clause-group with body $b = x$ at line 15 of Algorithm REVISEHORN, where again x was returned by Algorithm SHRINKEXAMPLE. Suppose for contradiction that b falsifies C_*, and that hypothesis clause-group body b_i covers C_*. Since $b = x$ falsifies C_* and b_i covers C_*, we have that $b_i \cap x$ falsifies C_*, so the **if** statement at line 6 should have directed the algorithm to use x to edit b_i as long as $b_i \cap x \subset b_i$. If instead $b_i \cap x = b_i$, then b_i falsifies C_*, and again by Lemma 10, it must be that b does not satisfy b_i's clause-group, contradicting the assumption that x satisfies the hypothesis. □

Theorem 14. *Algorithm* REVISEHORN *uses at most* $O(m^3e + m^4)$ *queries to revise a Horn sentence containing* m *clauses and needing* e *revisions.*

Proof. First, remember that, by Corollary 11, once a particular clause-group is added, it is never deleted. By Lemmas 9 and 13, the number of clause-groups is at most the number of clauses in the target formula. Thus in the worst case one clause-group C is introduced into the hypothesis for each target clause C_*.

Let us consider how many queries that one clause-group C can generate over the lifetime of the algorithm. Its creation required $O(m^2)$ queries for the call to SHRINKEXAMPLE for the negative counterexample, plus $O(m)$ in the main code of REVISEHORN.

Next, consider the manipulation of heads in the clause-group. There can be at most m heads introduced to a clause (plus **F**). By Lemma 12, each of them can be removed or moved exactly once. Each such edit uses $O(1)$ queries.

Finally, consider the use of negative counterexamples to edit the body of the clause-group C. By Lemma 7, each such negative counterexample may cost $O(m^2)$ queries. We will get our overall query bound by showing that the number of edits to the body of C is at most $O(m + e)$.

At any point in the run of the algorithm, body(C) falsifies (at least one) target clause C_*, by Lemma 9. By Lemma 8, the variables in body(C) fall into three categories:

1. Those in body(C_*) (which should not be deleted).
2. Variables that are heads of some initial theory clause.
3. Variables that are in the initial theory clause C_0 from which C_* is derived, but are not in C_*. That is, the variables that need the revision.

Now there are at most m heads of initial theory clauses, and there are at most e variables that need to be deleted.

This is not quite the whole proof, however. Lemma 9 says that body(C) must always falsify the body of some target clause, but it does not say that it must always be the same target clause.

A negative counterexample may cause body(C) to change which target clause body it falsifies. We now argue that this can happen only $m-1$ times, because once body(C) ceases to falsify a particular target clause, it can never again in its life falsify that target clause. This is so because the only way that the clause-group body could stop falsifying target clause C_* would be by having some variable in body(C_*) deleted from the clause-group, and once a variable is deleted from a clause-group body it is never put back in.

Moreover, the $m+e$ edits of body(C) accounted for in items 2 and 3 above are the total for the entire life of clause-group C, not just for the period while clause-group C is associated with one particular target clause. This is so because m is the total number of heads of initial theory clauses that might ever have to be deleted from C in its lifetime, and again, once one of those heads is deleted it is never replaced. Similarly, e is an upper bound on the total number of deletions to be made from all initial theory clauses, and once one of those variables is deleted, it is never replaced.

Since there are up to m hypothesis clause-groups, the total algorithm requires $O(em^3 + m^4)$ queries. $\square$

5.4. *A lower bound on revising Horn sentences*

In this subsection, we give a lower bound on the query complexity of revising Horn sentences. The argument shows that in general we cannot escape some dependence on the number of clauses in the initial formula. We give a Horn sentence where $\Omega(m)$ queries are required to make a single deletion revision. Note that we will not have to specify the target function in advance, because, as is usual with adversary arguments, we need only make sure that all our adversary's responses are consistent with some target function.

The technical argument is very similar to our lower bound on revising DNF [30], here transformed for the CNF form of Horn sentences.

Consider the variables $x_1, \ldots, x_n, y_1, \ldots, y_n$ and let $\varphi_n = c_1 \wedge \cdots \wedge c_n$, where, for $i = 1, \ldots, n$,

$$c_i = (x_1 \cdots x_{i-1} x_{i+1} \cdots x_n \wedge y_i \to \mathbf{F}).$$

Theorem 15. *The formula φ_n requires at least $n-1$ membership and equivalence queries to be revised, if each equivalence query must be a conjunction of Horn clauses, with each Horn clause body the revision of some body of a clause in φ_n, even if it is known that exactly one literal y_i is deleted.*

Proof. We describe how an adversary can answer the queries of any possible revision algorithm in a way that forces the revision algorithm to make the claimed number of queries. Let ψ_i be the formula obtained from φ_n by deleting the single occurrence of variable y_i. Initially any concept in $\Psi = \{\psi_1, \ldots, \psi_n\}$ is a possible target concept, and the adversary strategy that we describe will eliminate at most one concept from Ψ per query made by the revision algorithm. This implies the claimed lower bound.

Let us use ordered pairs (x, y) to denote truth assignments to the $2n$ variables, where the first component x will be the truth assignment to the x_i's and the second component the truth assignment to y_i's.

A membership query (x, y) is answered as follows. If x has at most $n-2$ bits that are 1, then $\mathrm{MQ}(x, y) = 1$. This does not eliminate any concepts from Ψ. If x has $n-1$ bits that are 1 with position $x_i = 0$, then $\mathrm{MQ}((x, y)) = \bar{y}_i$. If $y_i = 1$ then this does not eliminate any concept from Ψ. If $y_i = 0$ then ψ_i is eliminated from Ψ. If $x = \mathbf{1}$ then $\mathrm{MQ}((x, y)) = 0$. This does not eliminate any concept from Ψ.

Now consider an equivalence query $\mathrm{EQ}(\theta)$, where θ is a conjunction of Horn clauses, and for each clause C of θ, we have that $\mathrm{body}(C)$ is a revised version of the body of some clause of φ_n.

If θ contains any clause C with at most $n-2$ of the x_i's in it, then return the positive counterexample that has a 1 for every position of $\mathrm{body}(C)$, and 0's elsewhere. This does not eliminate any concept from Ψ.

If θ has no clause with at most $n-2$ of the x's, but contains at least one clause C_i with $\mathrm{body}(C_i)$ being all the x's except x_i (and no y), then return the positive counterexample that has a 1 for every position of $\mathrm{body}(C)$, and 0's elsewhere. This eliminates only concept ψ_i from Ψ.

The final possibility is that every clause in θ has exactly $n-1$ of the x's in it together with the corresponding y. (This case includes the case where $\theta = \varphi_n$.) In this case, return the negative counterexample $1^n 0^n$. This does not eliminate any concept from Ψ. □

6. Revising read-once formulas

In this section we present a revision algorithm for the class of read-once formulas, and lower bounds showing that the algorithm is close to optimal. In the first subsection we give some preliminaries for the revision algorithm. This is followed by the description of the algorithm, its analysis and a detailed example. The final subsection gives the lower bounds.

6.1. Sensitization, subformulas

Our revision algorithm uses the technique of *path sensitization* from fault analysis in switching theory (see, e.g., Kohavi [39]). Assume that we would like to revise the monotone read-once formula

$$\varphi = (\varphi_1 \vee \varphi_2) \wedge \varphi_3,$$

and let the target formula be

$$\psi = (\psi_1 \vee \psi_2) \wedge \psi_3,$$

where ψ is obtained from φ by replacing certain variables by constants. Consider the partial truth assignment α that fixes all the variables in φ_2 to 0, and all the variables in φ_3 to 1. This fixing of the variables is called *sensitizing* φ_1, and α is called the *sensitizing partial truth assignment* for φ_1. Form two vectors x_0 and x_1 by fixing the remaining variables to 0, respectively, to 1, and ask the membership queries $\mathrm{MQ}(x_0)$ and $\mathrm{MQ}(x_1)$.

There are three possibilities.

1. If $\mathrm{MQ}(x_1) = 0$, then it must be the case that *either* $\psi_1(\mathbf{1}) = 0$, in which case ψ_1 is identically 0, *or* $\psi_3(\mathbf{1}) = 0$, in which case the whole target formula is identically 0.
2. If $\mathrm{MQ}(x_0) = 1$, then it must be the case that *either* $\psi_1(\mathbf{0}) = 1$, in which case ψ_1 is identically 1, *or* $\psi_2(\mathbf{0}) = 1$, in which case ψ_2 is identically 1.
3. For the revision algorithm it is important to notice that we can also gain information in the third case, when $\mathrm{MQ}(x_0) = 0$ and $\mathrm{MQ}(x_1) = 1$. In this case we do not observe any "abnormality", but we can conclude that for every truth assignment y to the variables of ψ_1 it holds that $\psi_1(y) = \mathrm{MQ}(y, \alpha)$. Thus we can simulate membership queries to the subformula ψ_1 by membership queries to the target concept, and this enables the revision algorithm to proceed by recursion. Also note that in this case it is still possible that $\psi_2(\mathbf{1}) = 0$ and/or $\psi_3(\mathbf{0}) = 1$.

Now we give the general definition of a sensitizing partial truth assignment. Let φ' be a subformula of φ. Consider the binary tree representing φ, and let P be the path leading from the root of φ to the root of φ'. Then φ can be written as

$$\varphi = (\cdots(\varphi' \circ_r \varphi_r) \circ_{r-1} \cdots \circ_3 \varphi_3) \circ_2 \varphi_2) \circ_1 \varphi_1, \tag{1}$$

where $\varphi_1, \ldots, \varphi_r$ are the subformulas corresponding to the siblings of the nodes of P, and $\circ_1, \ldots, \circ_r$ are either $\wedge$ or $\vee$. In this representation we used the commutativity of $\wedge$ and $\vee$; in general φ' need not be a leftmost subformula of φ. Let ψ be obtained from φ by replacing certain variables by constants. Then, as in (1), we can write ψ as

$$\psi = (\cdots(\psi' \circ_r \psi_r) \circ_{r-1} \cdots \circ_3 \psi_3) \circ_2 \psi_2) \circ_1 \psi_1. \tag{2}$$

Definition 16. Let φ be a read-once formula with subformula φ'. Write φ as in Eq. (1). Let the sets of variables occurring in φ_i be X_i, and the set of variables occurring in φ' be Y. Since φ is read-once, these sets form a partition of $\{x_1, \ldots, x_n\}$. Now let α be the partial truth assignment that assigns 1 (respectively, 0) to every variable in X_i if $\circ_i$ is $\wedge$ (respectively, $\vee$), for every $i = 1, \ldots, r$. Then α is called the *partial truth assignment sensitizing* φ'.

Generalizing the remarks above, let α be the partial truth assignment sensitizing φ'. Form the truth assignments $x_0 = (\mathbf{0}, \alpha)$ (respectively $x_1 = (\mathbf{1}, \alpha)$) that extend α by assigning 0 (respectively 1) to the variables occurring in φ'. Now, if $\mathrm{MQ}(x_1) = 0$, then it follows by the monotonicity of ψ that either ψ' or a subformula ψ_i such that $\circ_i = \wedge$ is constant 0. In this case, the whole subformula corresponding to $(\cdots(\psi' \circ_r \psi_r) \circ_{r-1} \cdots \circ_{i-1} \psi_{i-1}) \circ_i \psi_i$ in the target must be constant 0; thus this whole subformula can be deleted and replaced by 0. The case is similar when $\mathrm{MQ}(x_0) = 1$. On the other hand, when $\mathrm{MQ}(x_1) = 1$ and $\mathrm{MQ}(x_0) = 0$, we can be sure that for any partial truth assignment y of the variables in ψ', we have $\psi'(y) = \mathrm{MQ}((y, \alpha))$. This means that ψ' is not part of a constant subformula. These remarks are summarized in the following lemma, which is used several times later on without mentioning it explicitly.

Lemma 17. (a) *Let φ be the initial formula, φ' be a subformula of φ, let ψ, ψ' be the target formula, respectively, its subformula corresponding to φ', and let α be the partial* *truth assignment sensitizing φ'. Then ψ' is part of a constant subformula if and only if MQ$(\mathbf{0}, \alpha) = 1$ or MQ$(\mathbf{1}, \alpha) = 0$. Otherwise $\psi'(y) = \mathrm{MQ}(y, \alpha)$ for every truth assignment y of the variables in φ'.*

(b) *If ψ' is a maximal constant subformula and $\circ_i$ is $\wedge$ (respectively $\vee$), then $\varphi_i(\mathbf{1}) = 1$ (respectively $\varphi_i(\mathbf{0}) = 0$) for every $i = 1, \ldots, r$.*

In the rest of this subsection we formulate some useful properties of subformulas. Two subformulas are *siblings* if the corresponding nodes in the tree representation are siblings. The next lemma follows directly from the definitions.

Lemma 18. *Two maximal constant subformulas cannot be siblings.*

The revision algorithm proceeds by finding maximal constant subformulas, thus it is important to know that identifying these is sufficient for learning.

Lemma 19. *Substitutions σ_1 and σ_2 are equivalent for formula φ if and only if the maximal constant subformulas of $\varphi\sigma_1$ and $\varphi\sigma_2$ are identical.*

Proof. If the maximal constant subformulas are identical, then after replacing them with the corresponding constants, one obtains the same formula. Thus the if direction of the lemma holds. For the only if direction, assume that σ_1 and σ_2 are equivalent for φ, but the maximal constant subformulas are not identical. There are two cases. The first case is when there is a subformula φ' of φ that turns into a maximal constant subformula in both $\varphi\sigma_1$ and $\varphi\sigma_2$, but $\varphi'\sigma_1 \equiv 0$ and $\varphi'\sigma_2 \equiv 1$. Let α be the partial truth assignment sensitizing φ'. Then $(\varphi\sigma_1)(\mathbf{1}, \alpha) = 0$, while $(\varphi\sigma_2)(\mathbf{1}, \alpha) = 1$, contradicting the assumption that σ_1 and σ_2 are equivalent. In the second case there is a subformula which is maximal constant for one substitution, but not for the other. Let φ' be a largest such subformula. We may assume w.l.o.g. that $\varphi'\sigma_1$ is a maximal constant subformula, which computes the constant 0, and $\varphi'\sigma_2$ is not part of a constant subformula. Then $\varphi\sigma_1(\mathbf{1}, \alpha) = 0$ and $\varphi\sigma_2(\mathbf{1}, \alpha) = 1$, again contradicting the assumption that σ_1 and σ_2 are equivalent. □

Corollary 20. *By finding a revision of the formula φ that has maximal constant subformulas identical to those of the target formula, we get a formula equivalent to the target formula.*

The following lemma can be proved by a simple algorithm that uses recursion on the structure of the formula φ.

Lemma 21. *Given a read-once formula φ and a constant c, one can find a substitution σ such that $\varphi\sigma = c$ and σ fixes a minimal number of variables, in polynomial time.*

Let φ be a read-once formula with subformula φ'. We say that φ' is an *approximately half-size subformula* of φ if it contains at least one-third and at most two-thirds of the variables in φ. It is a standard fact that such a subformula exists (see, e.g., Wegener [56]).

```
1: while $(x = \mathrm{EQ}(\varphi)) \neq$ "correct" do
2:     $\sigma = \textsc{FindConstant}(\varphi, x)$
3:     $\varphi = \varphi\sigma$
4: end while
```

Algorithm 4. Algorithm ReviseReadOnce(φ).

For example, any minimal subformula that contains at least one-third of the variables has this property.

If φ is a read-once formula with subformula φ', then the *φ'-partition* of a truth assignment x is (x_1, x_2), where x_1 contains the values in x for all the variables in φ', and x_2 contains the values in x for all the variables in φ that are not in φ'.

6.2. The revision algorithm

Now we formulate the main result of this section, for Algorithm ReviseReadOnce (Algorithm 4), which revises read-once formulas in the deletions-only model of revisions.

Theorem 22. *Algorithm* ReviseReadOnce *uses at most* $O(e \log n)$ *queries to revise a read-once formula containing n variables and needing e revisions.*

Proof. Algorithm ReviseReadOnce consists of a loop that checks whether the target has been found and if not calls FindConstant. In each call of FindConstant by ReviseReadOnce, we identify a maximal constant subformula of the target formula ψ, and we find a substitution that fixes this subformula to the appropriate constant value. The maximal constant subformula is then eliminated, thus the updated formula contains fewer variables. As the membership queries always refer to truth assignments to the original set of variables, the new membership queries have to assign some values to the eliminated variables as well. The construction implies that these variables are irrelevant, therefore their values can be arbitrary.

FindConstant, displayed as Algorithm 5, is a recursive procedure, which takes a formula φ and a counterexample x, and returns a substitution σ. The substitution fixes a subformula to a constant c. It always holds that the subformula is a maximal constant subformula computing the constant c in any representation of the target concept.[5] FindConstant works recursively, always focusing on a faulty subformula (i.e., a subformula which contains some variable(s) replaced by a constant) of the previous level's formula. This subformula may never be a *proper* subformula of a constant subformula—that is, it is part of a constant subformula if and only if it itself is a maximal constant subformula. We assume this property holds at the beginning of every recursion level, and we maintain it as we go deeper in the recursion. This guarantees that we eventually find

[5] In several places in the proof we will say that a property holds for any representation of the target concept. Notice that this must be true, as all the information used by the algorithm comes from membership and equivalence queries about the target, and the responses to such queries are independent of the particular representation.

```
1: if $\varphi$ has one variable then
2:     return substitution $\sigma$ fixing it to constant $\neg\varphi(x)$
3: end if
4: if MQ($\mathbf{0}$) == 1 or MQ($\mathbf{1}$) == 0 then
5:     return substitution $\sigma$ fixing $\varphi$ to the appropriate constant
6: end if
7: $\varphi' =$ an approximately half-size subformula of $\varphi$
8: $\alpha =$ the partial truth assignment sensitizing $\varphi'$
9: if (MQ($\mathbf{0}, \alpha$) == MQ($\mathbf{1}, \alpha$) == $c$) then
10:     return GROWFORMULA($\varphi, \varphi', c$)
11: else
12:     $(x_1, x_2) =$ the $\varphi'$-partition of $x$ and $x_2 = (x_{2,1}, \ldots, x_{2,r})$ corresponding
        to subformulas
13:     if MQ($x_1, \alpha$) $\neq \varphi'(x_1)$ then
14:         FINDCONSTANT($\varphi', x_1$)                // look in $\varphi'$
15:     else
16:         $i =$ FINDFORMULA($\varphi, \varphi', x$)
17:         FINDCONSTANT($\varphi_i, x_{2,i}$)
18:     end if
19: end if
```

Algorithm 5. The procedure FINDCONSTANT(φ, x).

a maximal constant subformula. Once such a subformula is found, we use Lemma 21 to return an appropriate substitution.

As we go deeper in the recursion, we will need the ability to ask membership queries concerning only a subformula of the target. Therefore, when we go to a lower recursion level with a subformula φ' of φ, we determine α, the partial truth assignment sensitizing φ'. This way, whenever a need for a membership query arises on the lower level for a truth assignment y, we need only ask MQ(y, α). Recursion only occurs when MQ$(\mathbf{0}, \alpha) = 0$ and MQ$(\mathbf{1}, \alpha) = 1$, thus we can be sure that MQ(y, α) is equal to the value of $\psi'(y)$, where ψ' is the subformula of the target formula corresponding to φ'. From now on, when talking about membership queries, we always assume that this technique is used. We write MQ(y) instead of MQ(y, α), where α is the partial truth assignment sensitizing the current subformula.

Now we give a detailed description of FINDCONSTANT, by explaining what it does on one level of the recursion: how it finds an appropriate faulty subformula, and how it maintains the counterexample x so it can be carried down into the next level as a counterexample. The correctness of the algorithm follows from this discussion directly. The complexity analysis requires only one point to be considered in detail. This is done in Lemma 23 at the end of the proof.

Lines 1–3: We check whether the current subformula φ consists of a single variable. If it does (say $\varphi = v_i$), then—since we know that φ is not a proper part of any constant subformula, but φ is faulty—we can be sure that φ is a maximal constant subformula; thus the substitution $v_i \to c$, where $c := \neg\varphi(x)$, will give the appropriate maximal constant subformula.

From now on we can assume that the input formula has more than one variable.

Lines 4–6: FINDCONSTANT examines whether $\mathrm{MQ}(\mathbf{0}) = 0$ and $\mathrm{MQ}(\mathbf{1}) = 1$. If not, then the whole subformula is identically true or false. Since φ has the property that it is not properly contained in a constant subformula, φ itself must be a maximal constant subformula.

Lines 7–8: We now know that φ is not part of a constant subformula. We determine an approximately half-size subformula φ' of φ, and its sensitizing partial truth assignment α.

Lines 9–10: We check if $\mathrm{MQ}(\mathbf{0}, \alpha) = \mathrm{MQ}(\mathbf{1}, \alpha) = c$. If that is the case, then $\mathrm{MQ}(y, \alpha) = c$ for any partial truth assignment y to the variables in φ'. Thus ψ' is a constant subformula, and so it is in a maximal constant subformula that is properly contained in ψ. At this point we do not perform any further recursive calls. The only task left is to find the node on the path P from the root of ψ to the root of ψ' that is the root of that maximal constant subformula. Procedure GROWFORMULA does this. As GROWFORMULA implements a simple binary search, we give only a brief description, without displaying its pseudocode. The procedure gets as input a read-once formula φ, a subformula φ', and a constant c such that $\mathrm{MQ}(\mathbf{0}, \alpha) = \mathrm{MQ}(\mathbf{1}, \alpha) = c$. Using $\mathrm{O}(\log n)$ membership queries it outputs a maximal subformula containing φ' such that the corresponding subformula is identical to the constant c in any representation of the target.

We now assume that $c = 1$; the case $c = 0$ is dual. Using the notation of Definition 16, let α_i for $i = 0, \ldots, r$ be the partial truth assignment that is identical to α for $X_1, \ldots, X_i$, leaves the variables in Y unassigned, and assigns 0 to all the other variables. Then $(\mathbf{0}, \mathbf{0}) = (\mathbf{0}, \alpha_0) \leqslant (\mathbf{0}, \alpha_1) \leqslant (\mathbf{0}, \alpha_2) \leqslant \cdots \leqslant (\mathbf{0}, \alpha_r) = (\mathbf{0}, \alpha)$, and it holds that $\mathrm{MQ}(\mathbf{0}, \alpha_0) = 0$ and $\mathrm{MQ}(\mathbf{0}, \alpha_r) = 1$.

Asking membership queries $\mathrm{MQ}(\mathbf{0}, \alpha_j)$, we can use binary search to find an i $(1 \leqslant i \leqslant r)$ such that $\mathrm{MQ}(\mathbf{0}, \alpha_{i-1}) = 0$ and $\mathrm{MQ}(\mathbf{0}, \alpha_i) = 1$. The only difference between the truth assignments $(\mathbf{0}, \alpha_{i-1})$ and $(\mathbf{0}, \alpha_i)$ is that the variables in X_i are off in $(\mathbf{0}, \alpha_{i-1})$ and they may be on in $(\mathbf{0}, \alpha_i)$. In fact, they must be on, as otherwise $(\mathbf{0}, \alpha_{i-1}) = (\mathbf{0}, \alpha_i)$. It also follows that $\circ_i$ is $\wedge$. Thus, on one hand, it must be the case that $\psi_i(\mathbf{0}) = 0$ and $\psi_i(\mathbf{1}) = 1$ in any representation of the target concept. On the other hand, it must be the case that the input to $\circ_i$ from its child on the path P is equal to 1 in both cases. As the variables in this subformula are all set to 0, this subformula must compute the constant 1 function. The inputs $(\mathbf{0}, \alpha_{i-1})$ and $(\mathbf{0}, \alpha_i)$ demonstrate that no larger subformula computes a constant function. Thus the subformula rooted at $\circ_{i-1}$ is a maximal constant subformula. This completes the discussion of the procedure GROWFORMULA.

Lines 11–12: If we get to line 11 then we know that ψ' is not part of a constant subformula, so we must continue the recursion to find one within ψ'. Using counterexample x, we form the φ'-partition of x in line 12. In the remainder of the procedure we find a faulty subformula that has at most two-thirds of the variables in φ.

Lines 13–14: Since α is the partial truth assignment sensitizing φ', we have $\varphi(x_1, \alpha) = \varphi'(x_1)$. Furthermore $\mathrm{MQ}(x_1, \alpha) = \psi'(x_1)$, because ψ' is not part of a constant subformula. If $\mathrm{MQ}(x_1, \alpha) \neq \varphi(x_1, \alpha)$ then $\varphi'(x_1) \neq \psi'(x_1)$, thus φ' contains a maximal constant subformula. Thus we can carry on finding some faulty parts that contribute to the faulty evaluation on x by the recursive call FINDCONSTANT(φ', x_1).

Lines 15–17: The only way we could get to this point is if $\mathrm{MQ}(x_1, \alpha) = \psi'(x_1) = \varphi'(x_1) = d$, and there are some faults in a subformula φ_i of φ for some $i \in \{1, 2, \ldots, r\}$. Let $x_2 = (x_{2,1}, x_{2,2}, \ldots, x_{2,r})$, where $x_{2,i}$ is the part of x_2 containing the variables in X_i.

Let y_i (respectively z_i) be the value computed at $\circ_i$ in φ (respectively ψ) on the input vector x, for $i = 1, \dots, r$. Furthermore let $y_{r+1} = d = z_{r+1}$. Then

$$y_i = y_{i+1} \circ_i \varphi_i(x_{2,i}), \quad \text{and} \quad z_i = z_{i+1} \circ_i \psi_i(x_{2,i})$$

for $i = 1, \dots, r$. Since x was a counterexample to $\mathrm{EQ}(\varphi)$, it holds that $y_1 = \varphi(x) \neq \psi(x) = z_1$.

Since $y_{r+1} = z_{r+1}$ and $y_1 \neq z_1$, there must be an i $(1 \leqslant i \leqslant r)$ for which $y_{i+1} = z_{i+1}$ but $y_i \neq z_i$. Now let us assume that we know this special i (the next paragraph describes the procedure FINDFORMULA for finding it). Then it follows that φ_i is faulty, and that $x_{2,i}$ is a counterexample to the equivalence of φ_i and ψ_i. This means that we can carry on with the search for the faulty subformula in φ_i. This can be done by a recursive call for FINDCONSTANT, using $x_{2,i}$ as the counterexample. As before, in the recursion we can simulate any assignment y to the variables in ψ_i by $\mathrm{MQ}(y, \tilde{\alpha})$, where $\tilde{\alpha}$ is the partial truth assignment sensitizing φ_i in φ (since φ' is not part of a constant subformula, neither is φ_i, thus the answer for this query will indeed give us the value $\psi_i(y)$).

The search for the appropriate index i is done by procedure FINDFORMULA using a weighted binary search as follows. The y_i values can be computed using φ without any queries. For the computation of the z_i, let β_i be the partial truth assignment that assigns $x_{2,j}$ to the variables in X_j for $j = i, \dots, r$ and otherwise is identical to α. Then $z_i = \mathrm{MQ}(x_1, \beta_i)$, since, as φ' is not contained in any constant subformula, there are no constant subformulas on the path $\circ_1, \dots, \circ_r$.

Let n_j denote the number of variables in φ_j, and define the *weight* of this subformula to be $w_j = n_{j-1} + n_j$ $(j = 2, \dots, r)$. In the binary search we use an interval $I = [a, b]$. Initially $a = 2$ and $b = r$, as we already know y_1, z_1, y_{r+1} and z_{r+1}. For a given I let $s = \sum_{j \in I} w_j$. In each step we have to find an index ℓ for which $\sum_{j=a}^{\ell-1} w_j < s/2 \leqslant \sum_{j=a}^{\ell} w_j$ (for this we don't need to ask any queries). We determine y_ℓ and z_ℓ (this can be done using one query). If $y_\ell \neq z_\ell$, then let $I = [\ell + 1, b]$, otherwise let $I = [a, \ell - 1]$. If I is nonempty, we compute s again, and continue the search. Otherwise the search is over, and if $y_\ell \neq z_\ell$, then ℓ is the i index we were looking for, otherwise it is $\ell - 1$. This completes the description and the analysis of the revision algorithm.

Since in each iteration we find a maximal constant subformula, and then we find a minimal substitution to fix the value computed by this subformula to the appropriate constant, it follows that FINDCONSTANT is called at most e times. The claimed complexity bound then follows from the following lemma.

Lemma 23. *When called by* LEARNREADONCE, *Procedure* FINDCONSTANT *uses at most a total of* $\mathrm{O}(\log n)$ *membership queries.*

Proof. The general idea of the proof is that the more queries consumed by FINDFORMULA, the smaller will be the recursive call to FINDCONSTANT.

Let us examine how procedure FINDFORMULA works. Let u be the number of variables in subformula φ on a level of the recursion. Since φ' is an approximately half-size subformula of φ, $\sum_{j=1}^{r} u_j \leqslant u \cdot (2/3)$; thus initially $s = \sum_{j \in I} w_j = (\sum_{j=2}^{r} 2 \cdot u_j) - w_1 - w_r < 2 \cdot u \cdot (2/3) = u \cdot (4/3)$. The value of s will reduce to less than its half in each iteration of the search, so after k queries s will be less than $1/2^k$ times its initial value.

Thus it will be at most $u \cdot (4/3) \cdot (1/2^k)$. We also know that if it is the index i that should be returned, then until the last query, the weight of φ_i or the weight of φ_{i-1} appears in s. But they both contain u_i, thus before the last query we have $s \geqslant u_i$. In summary, if we get the index i in t iterations, then we used t queries, and the number of variables in φ_i is $u_i \leqslant u \cdot (4/3) \cdot (1/2^{t-1}) = u/(3 \cdot 2^{t-3})$. Thus using $t \leqslant \log(u/u_i) + 3 - \log 3 < \log(u/u_i) + 2$ queries we managed to restrict the location of the faulty subformula to φ_i containing u_i variables. Thus on this recursion level we had to use fewer than $K + 2 + \log(u/u_i)$ queries, where K is the number of queries needed in lines 4, 9 and 13.

The other way to enter the next recursion level is through line 14, which does not need any additional queries above K. Furthermore at the bottom of recursion we need at most $\mathrm{O}(\log u)$ queries (lines 1–10).

Note that on every level of the recursion the size of the subformula is at most two-thirds of the size at the previous level. Thus, denoting the size of the formula on the ith level of recursion by m_i, we have at most $q = \log_{2/3} m_0$ levels of recursion, and on each level (excluding the final one) we use at most $K + 2 + \log(m_i/m_{i+1})$ queries. Adding them up, we get that in one run of FINDCONSTANT we use

$$\left(K + 2 + \log\frac{m_0}{m_1}\right) + \cdots + \left(K + 2 + \log\frac{m_{q-1}}{m_q}\right) + \mathrm{O}(\log m_q) = \mathrm{O}(\log m_0)$$

queries. □

The proof of Lemma 23 concludes the proof of the theorem. □

6.3. Example run of revision algorithm for read-once formulas

Here is a detailed example showing how the read-once revision algorithm works. Let the formula to be revised be

$$\varphi = ((y_1 \wedge y_2) \vee (y_3 \wedge y_4)) \wedge ((((y_5 \wedge y_6) \vee y_7) \wedge y_8) \vee y_9)$$

and the substitution giving the target formula be

$$\sigma = (y_3 \to 1, y_5, y_6, y_8 \to 0). \tag{3}$$

Thus the target concept is represented by the formula

$$\psi = ((y_1 \wedge y_2) \vee (1 \wedge y_4)) \wedge ((((0 \wedge 0) \vee y_7) \wedge 0) \vee y_9).$$

We start by asking the equivalence query $\mathrm{EQ}(\varphi)$. Let us assume that we receive the negative counterexample $x = 110011110$. In Procedure FINDCONSTANT, the membership queries $\mathrm{MQ}(\mathbf{0}) = 0$ and $\mathrm{MQ}(1) = 1$ bring us to line 7. At this point we find an approximately half-size subformula, for example

$$\varphi' = (y_1 \wedge y_2) \vee (y_3 \wedge y_4).$$

The corresponding subformula of the target is $\psi' = (y_1 \wedge y_2) \vee (1 \wedge y_4)$.

Now we form the sensitizing truth assignment α for φ', which in this case simply sets all variables not in φ' to 1, and we ask membership queries for $(\mathbf{0}, \alpha)$ and for $(\mathbf{1}, \alpha)$. The answer is $\mathrm{MQ}(\mathbf{0}, \alpha) = 0$ and $\mathrm{MQ}(\mathbf{1}, \alpha) = 1$, and thus we continue on line 12. We have

$x_1 = 1100$ and $x_2 = 11110$. By asking the membership query $\mathrm{MQ}(x_1, \alpha)$ we find that $\psi'(x_1) = 1$. Knowing φ, we can determine without asking any queries that $\varphi'(x_1) = 1$. As $\psi'(x_1) = \varphi'(x_1)$, it follows that the x_2 part of the counterexample is responsible for the disagreement between $\varphi(x)$ and $\psi(x)$. In this particular case, the variables in x_2 happen to induce a subformula of φ, and so FINDFORMULA does not need to do anything. We substitute 1 for φ'. Then $x_2 = 11110$ is a negative counterexample for the new target, which is the subformula ψ'' of the target corresponding to

$$\varphi'' = ((((y_5 \wedge y_6) \vee y_7) \wedge y_8) \vee y_9).$$

It is important to note that as $\psi''(y) = \psi(x_1, y)$, we can simulate membership queries to the new target by membership queries to the original target; thus we can continue the same procedure recursively.

As the subsequent iterations illustrate additional cases, we give further steps of the algorithm on the example. In the next call, which is FINDCONSTANT(φ'', x_2), we again get to line 7. The next half size subformula can be $y_5 \wedge y_6$. The sensitizing truth assignment for this subformula is 010. Now, the membership queries to $(00, 010)$ and $(11, 010)$ both return 0, indicating that either $y_5 \wedge y_6$ or some subformula containing it is turned into the constant 0. Thus we call GROWFORMULA, which asks the additional membership queries $\mathrm{MQ}(11, 110) = 0$ and $\mathrm{MQ}(11, 111) = 1$. This shows that

$$(((y_5 \wedge y_6) \vee y_7) \wedge y_8)$$

is a maximal constant 0 subformula in φ''. No further recursive calls are needed, we only need to compute the minimal number of variables that, when turned to 0, make the subformula identically 0. This can be achieved by the single substitution $y_8 \to 0$. Now we have completed one call of the procedure FINDCONSTANT by the main program.

The next call of FINDCONSTANT start with an equivalence query for the formula obtained by the substitution just found, that is,

$$\varphi''' = ((y_1 \wedge y_2) \vee (y_3 \wedge y_4)) \wedge y_9.$$

Let us assume that we receive the positive counterexample 000111111, which, restricted to the five variables in φ''', is 00011. We continue with the half size subformula $y_1 \wedge y_2$, which divides the counterexample into 00 and 011. The sensitizing partial truth assignment to the first half is 001. We find that $\mathrm{MQ}(00, 001) = 0$ and $\mathrm{MQ}(11, 001) = 1$, thus $y_1 \wedge y_2$ is not turned into a constant subformula. (Notice that our only membership oracle needs inputs from $\{0, 1\}^9$; fortunately, we may give any values to the "missing" variables.) The membership query $\mathrm{MQ}(00, 001) = 0$ tells us that the first half of the counterexample gives the same output in $y_1 \wedge y_2$ and in the corresponding subformula of the target. To recurse, we must find a *subformula* of φ''' that contains some constant subformula, but the three variables y_3, y_4, and y_9 do not induce a subformula of φ'''. This is achieved by the procedure FINDFORMULA.

In this case we need consider only the two subformulas $y_3 \wedge y_4$ and y_9, though in general there could be $\Omega(n)$ such subformulas, necessitating the binary search performed by FINDFORMULA. By definition, φ''' disagrees with the target on the counterexample, and we have just concluded that $y_1 \wedge y_2$ agrees with the counterexample. So, if subformula $(y_1 \wedge y_2) \vee (y_3 \wedge y_4)$ of φ''' disagrees with the corresponding subformula of the target, then the subformula containing a constant subformula must be $y_3 \wedge y_4$. Otherwise it is y_9. To test whether the subformula $(y_1 \wedge y_2) \vee (y_3 \wedge y_4)$ agrees with the target on the counterexample, we ask a membership query on an instance formed by setting y_1, y_2, y_3, and y_4 to the values they have in the counterexample, and setting the remaining variable (y_9) to the value it had in the sensitizing assignment for $y_1 \wedge y_2$. That, is we make the query $\mathrm{MQ}(00011) = 1$. Since $\varphi'''(00011) = 0$, which disagrees with the target, there must be a constant subformula in $y_3 \wedge y_4$, which is the input subformula for the next call to FINDCONSTANT.

That call will identify the substitution $y_3 \to 1$, and the next equivalence query to the formula

$$((y_1 \wedge y_2) \vee y_4) \wedge y_9$$

will finally identify the target concept. Notice that we have actually revised *fewer* variables than given in Eq. (3). The number of variables revised is as small as possible for obtaining the target concept.

6.4. Lower bounds on revising read-once formulas

We prove a lower bound to the query complexity of revising read-once formulas by giving an example of an n-variable read-once formula, for which $\Omega(e\log(n/e))$ equivalence and membership queries are required to find a distance e revision. If $e = O(n^{1-\varepsilon})$ for some fixed $\varepsilon > 0$, then this lower bound is of the same order of magnitude, as the upper bound provided by REVISEREADONCE. It is also shown that both types of queries are needed for efficient revision. There are n-variable read-once formulas for which at least $n/2$ equivalence queries are required in order to find a single revision. For membership queries we present an even stronger lower bound, which shows that at least $n - e$ membership queries may be necessary, if (instead of not using equivalence queries at all) one is allowed to use *fewer than e* equivalence queries. As REVISEREADONCE uses exactly e equivalence queries to find a distance e revision, this means that just by allowing one fewer equivalence query, the number of membership queries required becomes linear. Bshouty and Cleve and Bshouty et al. [16,17] give somewhat related constructions and tradeoff results for different query types.

Our first two lower bounds are based on read-once formulas of the form $\bigvee(x_i \wedge y_i)$, using a VC-dimension, respectively an adversary argument, and the third lower bound uses an adversary argument for the n-variable disjunction.

Theorem 24. *The complexity of revising read-once formulas in the deletion-only model is $\Omega(e\log\frac{n}{e})$, where n is the number of variables in the initial formula and e is the revision distance between the initial formula and the target formula.*

Proof. Let us assume that

$$n = 2me, \quad \text{where } m = 2^t.$$

We use variables $x_{i,j}$ and $y_{i,j}$, where $1 \leqslant i \leqslant e$ and $0 \leqslant j \leqslant m-1$. The initial formula is

$$\varphi_n = \bigvee_{i=1}^{e} \bigvee_{j=0}^{m-1} (x_{i,j} \wedge y_{i,j}).$$

Assume the x and y variables be arranged in respective $e \times m$ matrices called X and Y, respectively. We look at the class of revisions of φ_n where in each row of the matrix X exactly one variable is fixed to 1. Let the corresponding concept class be $\mathcal{C}_n$.

Lemma 25. VC-dim$(\mathcal{C}_n) \geqslant e \cdot t$.

Proof. For $1 \leqslant k \leqslant e$ and $1 \leqslant \ell \leqslant t$ let

$$(X_{k,\ell}, Y_{k,\ell})$$

be a truth assignment that consists of all 0's, with the exception of some positions in the kth row of the Y matrix: namely, those positions (k, j), where the ℓth bit of the binary representation of j is 1. Let the set of these assignments be S. We claim that S is shattered by $\mathcal{C}_n$.

Consider a subset $A \subseteq S$. For every k $(1 \leqslant k \leqslant e)$ let a_k be the t-bit number describing which truth assignments $(X_{k,\ell}, Y_{k,\ell})$ belong to A. (That is, the ℓth bit of a_k is 1 iff $(X_{k,\ell}, Y_{k,\ell}) \in A$.) We look at the revision φ_A for which it is the a_kth variable which is fixed to 1 in row k of the matrix X.

It remains to show that this revision classifies S in the required manner. If $(X_{k,\ell}, Y_{k,\ell}) \in A$, then bit ℓ of a_k is 1. By definition, $Y_{k,\ell}$ has a 1 at position (k, a_k). In φ_A, the variable x_{k,a_k} is fixed to 1. These observations imply that

$$\varphi_A(X_{k,\ell}, Y_{k,\ell}) = 1.$$

On the other hand, if $(X_{k,\ell}, Y_{k,\ell}) \notin A$, then bit ℓ of a_k is 0. The only 1 components of $(X_{k,\ell}, Y_{k,\ell})$ are in row k of the Y matrix: these are those positions (k, j), where the ℓth bit of the binary representation of j is 1. Position (k, a_k) is not one of those. Thus the corresponding x-variables are not fixed to 1 in φ_A, and as their value is 0, we get

$$\varphi_A(X_{k,\ell}, Y_{k,\ell}) = 0. \qquad \square$$

By introducing dummy variables if n is not of the right form, we get

$$\text{VC-dim}(\mathcal{C}_n) \geqslant e \left\lfloor \log \frac{n}{2e} \right\rfloor.$$

The theorem now follows from the general result that the VC-dim$(\mathcal{C}_n)$ provides a lower bound to the number of equivalence and membership queries required to learn $\mathcal{C}_n$ up to a constant factor, even if the equivalence queries are not required to be proper [11,42]. $\square$

The number of formulas within revision distance e of a given read-once formula is at most $2^e \cdot \binom{n}{e}$. Thus if we allow equivalence queries which are not necessarily proper, then by using the standard halving algorithm [41] one can learn a revision using $\log(2^e \cdot \binom{n}{e}) = \mathrm{O}(e \log n)$ many equivalence queries. We now show that such a result is not possible if the queries are required to be proper.

Theorem 26. *The complexity of revising read-once formulas in the deletion-only model with* proper equivalence queries *is at least $\lfloor n/2 \rfloor$, where n is the number of variables in the initial formula, even if we assume that a single revision occurs.*

Proof. We use the initial formula

$$\varphi_n = \bigvee_{i=1}^{\lfloor n/2 \rfloor} (x_{2i-1} \wedge x_{2i}). \tag{4}$$

Variables in the same conjunction are called *partners*. The revisions considered fix exactly one variable to 1. Let the formula obtained from φ_n by fixing x_j to 1 be φ_n^j, and let the class $\mathcal{C}'_n$ consist of the formulas φ_n^j. We describe an adversary strategy that forces every learner to use at least $\lfloor n/2 \rfloor$ equivalence queries.

It may be assumed that the hypotheses are consistent with the previous counterexamples, otherwise one of the previous counterexamples can be returned again. Let us assume that the learner asks an equivalence query EQ(ψ).

If both a variable and its partner is fixed to 1 in ψ (i.e., $\psi \equiv 1$), then return 0 as a *negative* counterexample. This does not rule out any concept from $\mathcal{C}'_n$

Otherwise, if some variable x_j is fixed to 0 in ψ then return the *positive* counterexample which is all 0's, except that x_j and its partner have value 1. Again, this does not rule out any concept from $\mathcal{C}'_n$.

Otherwise, if a variable x_j is fixed to 1 in ψ but its partner is not, then return the *negative* counterexample which is all 0's except that the partner of x_j has value 1. This rules out the formula φ_n^j.

Finally, there remains the case when ψ is the initial formula (ψ does not have to be the first query). In this case the adversary looks at the set of formulas φ_n^j which are not ruled out yet. If there are more formulas with j even (respectively, odd), then it returns the *positive* counterexample 101010... (respectively, 010101...). This rules out all the formulas φ_n^j with j odd (respectively, even), but it does not rule out any with j even (respectively, odd).

The last query eliminates at most $\lfloor n/2 \rfloor$ concepts from $\mathcal{C}'_n$, and all the other queries eliminate at most one concept. As long as there is more than one concept which is not ruled out, the learning process cannot terminate, and thus the lower bound follows. □

Now we present a lower bound for the case when only membership queries are allowed. Actually, we consider a more general scenario, where the learner is allowed to ask a limited number of equivalence queries. In particular, we assume that the learner is told in advance that the target is at revision distance e from the initial theory, and the number of equivalence queries allowed is at most $e - 1$.

Theorem 27. *The number of membership queries required for revising read-once formulas in the deletion-only model is at least $n - e$, where n is the number of variables in the initial formula, e is the revision distance between the initial formula and the target formula, assuming that the number of equivalence queries is fewer than e.*

Proof. We start from the initial formula $x_1 \vee \cdots \vee x_n$, and we consider the class $\mathcal{C}''_n$ of revisions which fix exactly e variables to 0. The adversary maintains a partition (D, U, Q) of the variables, where D stands for *deleted*, U stands for *undeleted* and Q stands for *?*. In the beginning $D = U = \emptyset$ and $Q = \{x_1, \dots, x_n\}$. In the course of the learning process it always holds that every concept from $\mathcal{C}''_n$ for which every variable in D is deleted, and no variable in U is deleted, is consistent with the previous answers. This implies that the learner cannot identify the target as long as $|D| < e$ and $|D \cup Q| > e$.

For a membership query MQ(x) we consider three cases. If $x_i = 1$ for some $i \in U$, then MQ$(x) = 1$ and the sets are not changed. Otherwise, if $x_i = 1$ for some $i \in Q$, then MQ$(x) = 1$, and the variable x_i is moved from Q to U. Otherwise, MQ$(x) = 0$ and the sets are not changed.

For an equivalence query EQ(ψ), we consider the following cases. If ψ is identically 1 (respectively, 0) then the all 0 (respectively, all 1) vector is given as a *negative* (respectively, *positive*) counterexample, and the sets are not changed. If there is a variable $x_i \in Q$ in ψ, then the vector which is all 0's except for x_i is given as a *negative* counterexample, and x_i is moved from Q to D. Otherwise, the characteristic vector of Q is returned as a *positive* counterexample, and the sets are not changed.

Initially $|D| = 0$, and $|D|$ is increased only by an equivalence query. As there can be fewer than e equivalence queries, $|D|$ is always less than e. Thus the learning process can only terminate by achieving $|D \cup Q| = e$. But initially $|D \cup Q| = n$, and its size is decreased only by a membership query. Therefore at least $n - e$ membership queries are needed. □

7. Revising parity in the general revision model

So far we considered only errors corresponding to the deletion of literals and terms. In practical theory revision algorithms one also has to deal with other types of errors such as the replacement of a variable with another one, or the addition of a variable or a term. Some of these error types are hard to define in general, and one has to be careful with their definition in particular cases (see, e.g., [10,40]). Replacements and additions appear to be harder to handle than deletions.

Let the variables $x_1, \dots, x_n$ be given. A parity function is the exclusive—or of a subset of the variables, or the complement of such a function. Thus a parity function can be specified by giving a $\vec{u} \in \{0, 1\}^n$, and an $a \in \{0, 1\}$, and writing the parity function φ as $\varphi(x) = \vec{u} \cdot x \oplus a$, where $\cdot$ denotes the mod 2 inner product of two vectors. Thus $\vec{u} \cdot x = \left(\sum_{i=1}^n u_i x_i\right) \bmod 2$.

Given a parity function, we now allow the deletion of a variable, the replacement of a variable by a constant or *another variable*, the *addition* of a variable, and for parity, also the addition of the constant 1. Given a parity function φ, we denote by $\mathcal{R}_\varphi$ the class of parity functions that can be obtained from φ using the enlarged set of revision operators, and we denote by $\mathcal{C}_\varphi$ the corresponding concept classes. Thus, $\mathcal{C}_\varphi$ is the class of all parity functions over the variables $x_1, \dots, x_n$. In these cases, unlike the rest of the paper, the concept classes do not depend on the initial formula. The only role played by the revision operator is to determine the performance metric: if the target concept can be obtained with a few revisions then we have to identify it with few queries.

Theorem 28. *There is a revision algorithm for parity functions in the general model of revisions, using* $\mathrm{O}(e \log n)$ *queries, where* e *is the revision distance between the initial and the target concept.*

Proof. Let $\varphi(x) = \vec{u}_\varphi \cdot x \oplus a$ be the parity function to be revised, and let $\psi(x) = \vec{u}_\psi \cdot x \oplus b$ be the target concept.

Since $\psi(\mathbf{0}) = b$, the value of b can be determined with the single equivalence query $\mathrm{MQ}(\mathbf{0})$. If $a \neq b$ then we change φ to its complement to achieve $a = b$, and if $a = b = 1$ then we reverse labels. Thus it may be assumed that $a = b = 0$.

The vectors $\vec{u}_\varphi$ and $\vec{u}_\psi$ differ in at most $2d$ bits.

The revision algorithm starts with the equivalence query φ. Let x be the counterexample received for this query. As $a = b = 0$, it holds that $x \neq \mathbf{0}$. Our goal now is to find a counterexample containing exactly one 1. Let x_1 and x_2, be obtained from x by switching off respectively the first or second half of the 1 components in x. Notice that $x = x_1 \oplus x_2$, and so

$$\begin{aligned} \varphi(x_1) \oplus \varphi(x_2) &= \varphi(x_1 \oplus x_2) = \varphi(x) \\ &\neq \psi(x) = \psi(x_1 \oplus x_2) \\ &= \psi(x_1) \oplus \psi(x_2), \end{aligned}$$

so exactly one of $\varphi(x_1) \neq \psi(x_1)$ and $\varphi(x_2) \neq \psi(x_2)$ hold. Thus exactly one of x_1 and x_2 is a counterexample, and one membership query will tell us which one is the counterexample. Continuing this process, a counterexample with a single 1 component can be found with $\mathrm{O}(\log n)$ membership queries. The variable corresponding to the 1 component must be one of the variables where φ and ψ differ. Hence ψ can be found by repeating this procedure $\mathrm{O}(e)$ times. □

8. Concluding remarks

Theory revision is important because when we already know a theory close to the desired theory, learning from scratch is wasteful, that is, needlessly expensive. This whole area has received relatively little theoretical study. We have presented here efficient algorithms for Horn and read-once formulas under the deletions-only revision model. We have given tight bounds on such revisions of read-once formulas. In addition, we have given an algorithm and tight bounds for the general revision of parity formulas. These results *prove* that in at least one formal model, there are efficient theory revision algorithms.

Additional results on revising various forms of DNF formulas may be found in our companion paper [30]. Work on revising Valiant's class of projective DNF functions [53] may be found in Sloan et al. [49].

The work presented here by no means exhausts the area of theory revision from the learning theory point of view. There are numerous open problems; for instance, the revision of threshold formulas, and the revision of Horn formulas in the general model of revision.

Acknowledgements

Preliminary versions of many of the results in this paper appeared in the three conference papers [28,29,50], and in a master's thesis of the third author [51].

We thank Jignesh Doshi for careful readings of preliminary versions of this manuscript. Robert Sloan wishes to thank NSF for providing him with some time for his own research while he was working there.

References

[1] H. Aizenstein, T. Hegedűs, L. Hellerstein, L. Pitt, Complexity theoretic results for query learning, Computational Complexity 7 (1998) 19–53.

[2] H. Aizenstein, L. Hellerstein, L. Pitt, Read-thrice DNF is hard to learn with membership and equivalence queries, in: Proc. of the 33rd Symposium on the Foundations of Computer Science, Pittsburgh, PA, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 523–532.

[3] D. Angluin, Learning propositional Horn sentences with hints, Technical Report YALEU/DCS/RR-590, Department of Computer Science, Yale University, New Haven, CT, 1987.

[4] D. Angluin, Learning regular sets from queries and counterexamples, Inform. Comput. 75 (2) (1987) 87–106.

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

[6] D. Angluin, Negative results for equivalence queries, Machine Learning 5 (1990) 121–150.

[7] D. Angluin, M. Frazier, L. Pitt, Learning conjunctions of Horn clauses, Machine Learning 9 (1992) 147–164.

[8] D. Angluin, L. Hellerstein, M. Karpinski, Learning read-once formulas with queries, J. ACM 40 (1) (1993) 185–210.

[9] D. Angluin, M. Kharitonov, When won't membership queries help?, J. Comput. System Sci. 50 (2) (1995) 336–355. Earlier version appeared 23rd STOC, 1991.

[10] S. Argamon-Engelson, M. Koppel, Tractability of theory patching, J. Artificial Intelligence Res. 8 (1998) 39–65.

[11] P. Auer, P.M. Long, Structural results about on-line learning models with and without queries, Machine Learning 36 (3) (1999) 147–181.

[12] A. Blum, L. Hellerstein, N. Littlestone, Learning in the presence of finitely or infinitely many irrelevant attributes, J. Comput. Syst. Sci. 50 (1) (1995) 32–40. Earlier version in: Proc. of the Fourth Annual Workshop on Computational Learning Theory (COLT 1991), Santa Cruz, CA, 1991, pp. 157–166.

[13] A. Blum, S. Rudich, Fast learning of k-term DNF formulas with queries, J. Comput. System Sci. 51 (3) (1995) 367–373.

[14] N. Bshouty, Exact learning Boolean function via the monotone theory, Inform. Comput. 123 (1995) 146–153.

[15] N. Bshouty, L. Hellerstein, Attribute-efficient learning in query and mistake-bound models, J. Comput. System Sci. 56 (3) (1998) 310–319.

[16] N.H. Bshouty, R. Cleve, On the exact learning of formulas in parallel, in: Proc. of the 33rd Symposium on the Foundations of Computer Science, Pittsburgh, PA, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 513–522.

[17] N.H. Bshouty, S.A. Goldman, T.R. Hancock, S. Matar, Asking questions to minimize errors, J. Comput. System Sci. 52 (2) (1996) 268–286. Earlier version in: Proc. of the Sixth Annual ACM Conference on Computational Learning Theory (COLT 1993), Santa Cruz, CA, 1993, pp. 41–50.

[18] L. Carbonara, D. Sleeman, Effective and efficient knowledge base refinement, Machine Learning 37 (2) (1999) 143–181.

[19] P. Damaschke, Adaptive versus nonadaptive attribute-efficient learning, Machine Learning 41 (2) (2000) 197–215.

[20] L. De Raedt, Logical settings for concept-learning, Artificial Intelligence 95 (1997) 187–201.

[21] R. Dechter, J. Pearl, Structure identification in relational data, Artificial Intelligence 58 (1992) 237–270.

[22] A. Dhagat, L. Hellerstein, PAC learning with irrelevant attributes, in: Proc. of the 35rd Annual Symposium on Foundations of Computer Science, Santa Fe, NM, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 64–74.

[23] W.F. Dowling, J.H. Gallier, Linear-time algorithms for testing the satisfiability of propositional Horn formulae, J. Logic Programming 1 (1984) 267–284.

[24] R. Feldman, Probabilistic revision of logical domain theories, PhD Thesis, Cornell University, Ithaca, NY, 1993.

[25] M. Frazier, Matters Horn and other features in the computational learning theory landscape: The notion of membership, PhD Thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, Technical Report UIUCDCS-R-94-1858, 1994.

[26] M. Frazier, L. Pitt, Learning from entailment: An application to propositional Horn sentences, in: Proc. Tenth International Conf. Machine Learning, Amherst, MA, Morgan Kaufmann, San Mateo, CA, 1993, pp. 120–127.

[27] O. Goldreich, S. Goldwasser, S. Micali, How to construct random functions, J. ACM 33 (4) (1986) 792–807.

[28] J. Goldsmith, R.H. Sloan, More theory revision with queries (extended abstract), in: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, Portland, OR, May 21–23, 2000, ACM Press, New York, 2000, pp. 441–448.

[29] J. Goldsmith, R.H. Sloan, B. Szörényi, G. Turán, Improved algorithms for theory revision with queries, in: Proc. 13th Annual Conference on Computational Learning Theory, Palo Alto, CA, Morgan Kaufmann, San Francisco, CA, 2000, pp. 236–247.

[30] J. Goldsmith, R.H. Sloan, G. Turán, Theory revision with queries: DNF formulas, Machine Learning 47 (2/3) (2002) 257–295.

[31] R. Greiner, The complexity of revising logic programs, J. Logic Programming 40 (1999) 273–298.

[32] R. Greiner, The complexity of theory revision, Artificial Intelligence 107 (1999) 175–217.

[33] V. Gurvich, On repetition-free Boolean functions, Uspekhi Mat. Nauk 32 (1) (1977) 183–184 (in Russian).

[34] A. Horn, On sentences which are true on direct unions of algebras, J. Symbolic Logic 16 (1951) 14–21.

[35] M. Karchmer, N. Linial, I. Newman, M. Saks, A. Wigderson, A combinatorial characterization of read-once formulae, Discrete Math. 114 (1993) 275–282.

[36] M. Kearns, M. Li, L. Pitt, L. Valiant, On the learnability of Boolean formulae, in: Proc. 19th Annual ACM Symp. Theory of Computing (STOC 1987), New York, ACM Press, New York, 1987, pp. 285–294.

[37] M. Kearns, L. Valiant, Cryptographic limitations on learning Boolean formulae and finite automata, J. ACM 41 (1) (1994) 67–95.

[38] H. Kleine Buning, T. Lettmann, Propositional Logic: Deduction and Algorithms, Cambridge University Press, Cambridge, 1999.

[39] Z. Kohavi, Switching and Finite Automata Theory, second edition, McGraw-Hill, New York, 1978.

[40] M. Koppel, R. Feldman, A.M. Segre, Bias-driven revision of logical domain theories, J. Artificial Intelligence Res. 1 (1994) 159–208.

[41] N. Littlestone, Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm, Machine Learning 2 (4) (1988) 285–318.

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

[43] J.A. Makowsky, Model theory and computer science: An appetizer, in: S. Abramsky, D.M. Gabbay, T.S.E. Maibaum (Eds.), Handbook of Logic in Computer Science, vol. 1 (Background: Mathematical Structures), Oxford University Press, Oxford, 1992, pp. 763–814.

[44] J.C.C. McKinsey, The decision problem for some classes without quantifiers, J. Symbolic Logic 8 (1943) 61–76.

[45] R.J. Mooney, A preliminary PAC analysis of theory revision, in: T. Petsche (Ed.), Computational Learning Theory and Natural Learning Systems, vol. III: Selecting Good Models, MIT Press, Cambridge, MA, 1995, pp. 43–53, Chapter 3.

[46] D. Mundici, Functions computed by monotone Boolean formulas with no repeated variables, Theoret. Comput. Sci. 66 (1989) 113–114.

[47] D. Ourston, R.J. Mooney, Theory refinement combining analytical and empirical methods, Artificial Intelligence 66 (1994) 273–309.

[48] K. Pillaipakkamnatt, V. Raghavan, Read-twice DNF formulas are properly learnable, in: Computational Learning Theory: EuroColt '93, in: The Institute of Mathematics and its Applications Conference Series, vol. 53, Oxford University Press, Oxford, 1994, pp. 121–132.

[49] R.H. Sloan, B. Szörényi, G. Turán, Projective DNF formulae and their revision, in: Learning Theory and Kernel Machines, 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, August 24–27, 2003, Proceedings, in: Lecture Notes in Artificial Intelligence, vol. 2777, Springer, Berlin, 2003, pp. 625–639.

[50] R.H. Sloan, G. Turán, On theory revision with queries, in: Proc. 12th Annual Conf. on Computational Learning Theory, Santa Cruz, CA, ACM Press, New York, 1999, pp. 41–52.

[51] B. Szörényi, Revision algorithms in computational learning theory, Master's Thesis, Dept. of Computer Science, University of Szeged, 2000 (in Hungarian).

[52] G.G. Towell, J.W. Shavlik, Knowledge-based artificial neural networks, Artificial Intelligence 70 (1/2) (1994) 119–165.

[53] L.G. Valiant, Projection learning, Machine Learning 37 (2) (1999) 115–130.

[54] M.H. Van Emden, R.A. Kowalski, The semantics of predicate logic as a programming language, J. ACM 23 (1976) 733–742.

[55] V.N. Vapnik, A.Y. Chervonenkis, On the uniform convergence of relative frequencies of events to their probabilities, Theor. Probab. Appl. 16 (2) (1971) 264–280.

[56] I. Wegener, The Complexity of Boolean Functions, Wiley-Teubner, 1987.

[57] P.H. Winston, T.O. Binford, B. Katz, M. Lowry, Learning physical descriptions from functional definitions, examples, and precedents, in: Proc. Natl. Conf. on Artificial Intelligence (AAAI-83), Washington, DC, 1983, pp. 433–439.

[58] S. Wrobel, Concept Formation and Knowledge Revision, Kluwer, Dordrecht, 1994.

[59] S. Wrobel, First order theory refinement, in: L. De Raedt (Ed.), Advances in ILP, IOS Press, Amsterdam, 1995, pp. 14–33.