New Revision Algorithms

Judy Goldsmith[1]*, Robert H. Sloan[2]**, Balázs Szörényi[3], and György Turán[2,3,4]***

[1] University of Kentucky, Lexington, KY 40506-0046, USA,
goldsmit@cs.uky.edu.
[2] University of Illinois at Chicago, Chicago, IL 60607, USA,
sloan@uic.edu,
WWW home page: http://www.cs.uic.edu/~sloan
[3] Hungarian Academy of Sciences and University of Szeged, Research Group on Artificial Intelligence, Szeged, Hungary,
szorenyi@rgai.hu
[4] gyt@uic.edu

Abstract. A revision algorithm is a learning algorithm that identifies the target concept, starting from an initial concept. Such an algorithm is considered efficient if its complexity (in terms of the resource one is interested in) is polynomial in the syntactic distance between the initial and the target concept, but only polylogarithmic in the number of variables in the universe. We give efficient revision algorithms in the model of learning with equivalence and membership queries. The algorithms work in a general revision model where both deletion and addition type revision operators are allowed. In this model one of the main open problems is the efficient revision of Horn sentences. Two revision algorithms are presented for special cases of this problem: for depth-1 acyclic Horn sentences, and for definite Horn sentences with unique heads. We also present an efficient revision algorithm for threshold functions.

1 Introduction

Efficient learnability has been studied from many different angles in computational learning theory for the last two decades, for example, in both the PAC and query learning models, and by measuring complexity in terms of sample size, the number of queries or running time. Attribute-efficient learning algorithms are required to be efficient (polynomial) in the number of relevant variables, and "super-efficient" (polylogarithmic) in the total number of variables [1, 2]. It is argued that practical and biologically plausible learning algorithms need to be attribute efficient.

A related notion, *efficient revision algorithms*, originated in machine learning [3–6], and has received some attention in computational learning theory as well.

* Partially supported by NSF grants CCR-0100040 and ITR-0325063.
** Partially supported by NSF grant CCR-0100336.
*** Partially supported by NSF grant CCR-0100336.

A revision algorithm is applied in a situation where learning does not start from scratch, but there is an initial concept available, which is a reasonable approximation of the target concept. The standard example is an initial version of an expert system provided by a domain expert. The efficiency criterion in this case is to be efficient (polynomial) in the *distance* from the initial concept to the target (whatever distance means; we get back to this in a minute), and to be "super-efficient" (polylogarithmic) in the total size of the initial formula. Again, it is argued that this is a realistic requirement, as many complex concepts can only be hoped to be learned efficiently if a reasonably good initial approximation is available. The notion of distance usually considered is a syntactic one: the number of edit operations that need to be applied to the initial representation in order to get a representation of the target. The particular edit operations considered depend on the concept class. Intuitively, attribute-efficient learning is a special case of efficient revision, when the initial concept has an empty representation. In machine learning, the study of revision algorithms is referred to as theory revision; detailed references to the literature are given in Wrobel's overviews of theory revision [7, 8] and also in our recent papers [9, 10].

The theoretical study of revision algorithms was initiated by Mooney [11] in the PAC framework. We have studied revision algorithms in the model of learning with equivalence and membership queries [9, 10] and in the mistake-bound model [12].

It is a general observation both in practice and in theory that those edit operations which delete something from the initial representation are easier to handle than those which add something to it. We have obtained efficient revision algorithms for monotone DNF with a bounded number of terms when both deletion and addition type revisions are allowed, but for the practically important case of Horn sentences we found an efficient revision algorithm only for the deletions-only model. We also showed that efficient revision of general (or even monotone) DNF is not possible, even in the deletions-only model. Finding an efficient revision algorithm for Horn sentences with a bounded number of terms in the general revision model (deletions and additions) emerged as perhaps the main open problem posed by our previous work on revision algorithms. The work presented here extends that of Doshi [13], who gave a revision algorithm for "unique explanations", namely depth-1, acyclic Horn formulas with unique, non-**F**, and unrevisable heads. Horn revision also happens to be the problem that most practical theory revision systems address. It is to be noted here that the notions of learning and revising Horn formulas are open to interpretation, as discussed in [10]; the kind of learnability result that we wish to extend to revision in this paper is Angluin et al.'s for propositional Horn sentences [14].

It seems to be an interesting general question whether attribute-efficiently learnable classes can also be revised efficiently. The monotone DNF result mentioned above shows that the answer is negative in general. We gave a positive answer for parity functions in [9] and for projective DNF in [12]. Projective DNF is a class of DNF introduced by Valiant [15], as a special case of his projective learning model, and as part of a framework to formulate expressive and biologically plausible learning models. In biological terms revision may be relevant for learning when some information is hard-wired from birth; see, e.g., Pinker [16] for recent arguments in favor of hereditary information in the brain.

Valiant showed that projective DNF are attribute-efficiently learnable in the mistake-bound model, and we extended his result by showing that they are efficiently revisable. Our algorithm was based on showing that a natural extension of the Winnow algorithm is in fact an efficient revision algorithm for disjunctions even in the presence of attribute errors.

Valiant's related models [17, 18] also involve threshold functions, and as threshold functions are also known to be attribute-efficiently learnable, this raises the question whether threshold functions can be revised efficiently. Threshold functions (also called Boolean threshold functions or zero-one threshold functions in the literature) form a much studied concept class in computational learning theory. Winnow is an attribute-efficient mistake-bounded learning algorithm [19]. Attribute-efficient proper query learning algorithms are given in Uehara et al. [20] and Hegedüs and Indyk [21]. Further related results are given in [22–25].

Results In this paper we present results for the two revision problems outlined above: the revision of Horn sentences and threshold functions, in the general revision model allowing both deletions and additions (more precise definitions are given in Section 2). We use the model of learning with membership and equivalence queries.

For Horn sentences, we show that one can revise two subclasses of Horn sentences with respect to both additions and deletions of variables. The new algorithms make use of our previous, deletions-only revision algorithm for Horn sentences [10] and new techniques which could be useful for the general question as well.

The first of the two classes is depth-1 acyclic Horn sentences. The class of acyclic Horn sentences was introduced by Angluin [26] in a paper that presented the first efficient learning algorithm for a class of Horn sentences, and was studied from the point of view of minimization and other computational aspects (see, e.g., [27]), and in the context of predicate logic learning [28]. We consider the subclass of depth-1 acyclic Horn sentences, where, in addition to assuming that the graph associated with the Horn sentence is acyclic, we also require this graph to have depth 1. One of our main results, Theorem 1, shows that this class can be revised using $O(dist(\varphi, \psi) \cdot m^3 \cdot \log n)$ queries, where n is the number of variables, φ is the m-clause initial formula, ψ is the target formula, and *dist* is the revision distance, which will be defined formally in Section 2.

We also give a revision algorithm for definite Horn sentences with unique heads, meaning that no variable ever occurs as the head of more than one Horn clause. For this class, we revise with query complexity $O(m^4 + dist(\varphi, \psi) \cdot (m^3 + \log n))$, where again φ is the initial formula and ψ is the target function (Theorem 2).

For threshold functions we give a revision algorithm using $O(dist(\varphi, \psi) \cdot \log n)$ queries (Theorem 3). In this algorithm the pattern mentioned above is reversed, and it turns out to be easier to handle additions than deletions. It is also shown that both query types are necessary for efficient revision, and that the query complexity of the algorithm is essentially optimal up to order of magnitude. Another interesting point is that the natural extension of Winnow mentioned above does not work in this more general context.

Organization of paper Preliminaries are given in Section 2, Horn formula revisions in Section 3, threshold functions in Section 4. Due to space constraints, complete proofs are deferred to the full version of the paper.

2 Preliminaries

We use standard notions from propositional logic such as variable, literal, term (or conjunction), clause (or disjunction), etc. The set of variables for n-variable formulas and functions is $X_n = \{x_1, \ldots, x_n\}$. (In this paper, n will always be the total number of variables.) *Instances* or *vectors* are elements $\mathbf{x} \in \{0,1\}^n$. When convenient we treat $\mathbf{x}$ as a subset of $[n]$ or X_n, corresponding to the components, resp. the variables, which are set to true in $\mathbf{x}$. Given a set $Y \subseteq [n] = \{1, \ldots, n\}$, we write $\chi_Y = (\alpha_1, \ldots, \alpha_n) \in \{0,1\}^n$ for the characteristic vector of Y. We write $\mathbf{x} = (x_1, \ldots, x_n) \leq \mathbf{y} = (y_1, \ldots, y_n)$ if $x_i \leq y_i$ for every $i = 1, \ldots, n$.

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 body(c) and head(c) for the body and head of c, respectively. A clause with an unnegated variable is called *definite* (or positive). We will consider clauses with no unnegated variables to have head $\mathbf{F}$, and will sometimes write them as $(\textit{body} \rightarrow \mathbf{F})$. A *Horn sentence* is a conjunction of Horn clauses. A Horn sentence is definite if all its clauses are definite. A Horn sentence has *unique heads* if no two clauses have the same head.

We define the *graph* of a Horn sentence to be a directed graph on the variables together with $\mathbf{T}$ and $\mathbf{F}$, with an edge from variable u to variable v (resp. $\mathbf{F}$ iff there is a clause with head v (resp. $\mathbf{F}$) having u in its body, and an edge from $\mathbf{T}$ to variable v if there is a clause consisting solely of variable v. A Horn sentence is *acyclic* if its graph is acyclic; the *depth* of an acyclic Horn sentence is the maximum path length in its graph [26].

For example, the Horn sentence

$$(x_1 \wedge x_2 \rightarrow x_3) \wedge (x_1 \wedge x_4 \rightarrow x_5) \wedge (x_4 \wedge x_6 \rightarrow \mathbf{F})$$

is depth-1 acyclic. Its graph has the edges (x_1, x_3), (x_2, x_3), (x_1, x_5), (x_4, x_5), $(x_4, \mathbf{F})$, and $(x_6, \mathbf{F})$ and this graph is acyclic with depth 1.

If $\mathbf{x}$ satisfies the body of Horn clause c, considered as a term, we say $\mathbf{x}$ *covers* c. Notice that $\mathbf{x}$ *falsifies* c if and only if $\mathbf{x}$ covers c and head$(c) \notin \mathbf{x}$. (By definition, $\mathbf{F} \notin \mathbf{x}$.)

For Horn clause body b (or any monotone term) and vector $\mathbf{x}$, we use $b \cap \mathbf{x}$ for the monotone term that has those variables of b that correspond to 1's in $\mathbf{x}$. As an example, $x_1 x_4 \cap 1100 = x_1$.

An n-variable *threshold function* TH_U^t is specified by a set $U \subseteq [n]$ and a *threshold* $0 \leq t \leq n$, such that for a vector $\mathbf{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ it holds that $\mathrm{TH}_U^t(\mathbf{x}) = 1$ iff at least t of the variables with subscripts in U are set to 1 in $\mathbf{x}$. In other words, $\mathrm{TH}_U^t(\mathbf{x}) = 1$ iff $\sum_{i=1}^n \alpha_i x_i \geq t$, where $\chi_U = (\alpha_1, \ldots, \alpha_n)$. We say that S is a *positive* (resp., *negative*) set if χ_S is a positive (resp., negative) example of the target threshold function. (As the number of variables is clear from the context, we do not mention it in the notation.) Note that for every non-constant threshold function its set of relevant variables and its threshold are well defined, thus every non-constant function has a unique representation. The variables with indices in U (resp., outside of U) are the *relevant* (resp., *irrelevant*) variables of TH_U^t. As noted in the introduction, functions of this type are also called Boolean threshold functions and 0-1-threshold functions, in order to distinguish them from the more general kind of threshold functions where the coefficients α_i can be arbitrary reals. We simply call them threshold functions, as we only consider this restricted class.

We use the standard model of membership and equivalence queries (with counterexamples), denoted by MQ and EQ [29]. In an equivalence query, the learning algorithm proposes a *hypothesis*, a concept h, and the answer depends on whether $h \equiv c$, 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 $\mathbf{x}$ such that $c(\mathbf{x}) \neq h(\mathbf{x})$.

2.1 Revision

The *revision distance* between a formula φ and a concept C is defined to be the minimum number of applications of a specified set of syntactic revision operators to φ needed to obtain a formula for C. The revision operators may depend on the concept class one is interested in. Usually, a revision operator can either be *deletion-type* or *addition-type*.

For disjunctive or conjunctive normal forms, the deletion operation can be formulated as *fixing an occurrence of a variable* in the formula to a constant. In the *general model*, studied in this paper, we also allow additions. The addition operation is to *add a new literal to one of the terms or clauses of the formula.* (Adding a new literal to open up a new clause or term would be an even more general addition-type operator, which we have not considered so far.) In the case of Horn sentences the new literals must be added to the body of a clause.

In the case of threshold functions, deletions mean deleting a relevant variable and additions mean adding a new relevant variable. In the *general model* for this class we also allow the *modification of the threshold*. We consider the modification of the threshold by any amount to be a single operation (as opposed to changing it by one); as we are going to prove upper bounds, this only makes the results stronger. Thus, for example, the revision distance between $\varphi = \mathrm{TH}^1_{\{x_1,x_2,x_4\}}$ and $\mathrm{TH}^3_{\{x_1,x_2,x_3,x_5\}}$ is 4 in the general model.

We use $dist(\varphi, \psi)$ to denote the revision distance from φ to ψ whenever the revision operators are clear from context. In general, the distance is not symmetric.

A *revision algorithm* for a formula φ has access to membership and equivalence oracles for an unknown target concept and must return some representation of the target concept. Our goal is to find revision algorithms whose query complexity is polynomial in $d = dist(\varphi, \psi)$, but at most *polylogarithmic* in n, the number of variables in the universe. Dependence on other parameters may depend on the concept class. For DNF (resp. CNF) formulas, we will allow polynomial dependence on the number of terms (resp. clauses) in φ; it is impossible to do better even for arbitrary monotone DNF in the deletions-only model of revision [9].

We state only query bounds in this paper; all our revision algorithms are computable in polynomial time, given the appropriate oracles.

2.2 Binary search for new variables

Many of our revision algorithms use a kind of binary search, often used in learning algorithms involving membership queries, presented as Algorithm 1. The starting points of the binary search are two instances, a negative instance **neg** and a positive instance **pos** such that **neg** $\leq$ **pos**. The algorithm returns two items: the first is a set of variables that when added to **neg** make a positive instance; the second is a variable that is critical in the sense that the first component plus **neg** becomes a negative instance if that variable is turned off.

Algorithm 1 BinarySearch(**neg**, **pos**).

```
Require: MQ(neg) == 0 and MQ(pos) == 1 and neg ≤ pos
 1: neg_0 := neg
 2: while neg and pos differ in more than 1 position do
 3:    Partition pos \ neg into approximately equal-size sets $d_1$ and $d_2$.
 4:    Put mid := neg with positions in $d_1$ switched to 0
 5:    if MQ(mid) == 0 then
 6:       neg := mid
 7:    else
 8:       pos := mid
 9: $v$ := the one variable on which pos and neg differ
10: return ((pos \ neg_0), $v$)
```

3 Revising Horns

In this section we give algorithms for revising two different classes of Horn sentences when addition of new variables into the bodies is allowed as well as deletion of variables.

3.1 Depth-1 acyclic Horn sentences

We show here how to revise depth-1 acyclic Horn sentences. Depth-1 acyclic Horn sentences are precisely those where variables that occur as heads never occur in the body of any clause. Notice that such formulas are a class of unate CNF. Previously we gave a revision algorithm for unate DNF (which would dualize to unate CNF) that was exponential in the number of clauses [9]. Here we give an algorithm for an important subclass of unate CNF that is polynomial in the number of clauses.

The general idea of the algorithm is to maintain a one-sided hypothesis, in the sense that all equivalence queries using the hypothesis must return negative counterexamples until the hypothesis is correct.

Each negative counterexample can be associated with one particular head of the target clause, or else with a headless target clause. We do this with a negative counterexample $\mathbf{x}$ as follows.

For a head variable v and instance $\mathbf{x}$, we will use the notation $\mathbf{x}^v$ to refer to $\mathbf{x}$ modified by setting all head variables *other than* v to 1. Note that $\mathbf{x}^v$ cannot falsify any clause with a head other than v. Since v will normally be the head of a Horn clause and we use $\mathbf{F}$ to denote the "head" of a headless Horn clause, we will use $\mathbf{x}^{\mathbf{F}}$ to denote $\mathbf{x}$ modified to set *all* head variables to 1.

The algorithm begins with an assumption that the revision distance from the initial theory to the target theory is e. If the revision fails, then e is doubled and the algorithm is repeated. Since the algorithm is later shown to be linear in e, this series of attempts does not affect the asymptotic complexity. We give a brief overview of the algorithm, followed by somewhat more detail, and the pseudocode, as Algorithm 2.

We maintain a hypothesis that is, viewed as the set of its satisfying vectors, always a superset of the target. Thus each time we ask an equivalence query, if we have not found the target, we get a negative counterexample $\mathbf{x}$. Then the first step is to ask a membership query on $\mathbf{x}$ modified to turn on *all* of the head positions. If that returns 0, then the modified $\mathbf{x}$ must falsify a headless target clause. Otherwise, for each head position h that is 0 in the original $\mathbf{x}$, ask a membership query on $\mathbf{x}^h$. We stop when the first such membership query returns 0; we know that $\mathbf{x}$ falsifies a clause with head h. In our pseudocode, we refer to the algorithm just described as Associate.

Once a negative counterexample is associated with a head, we first try to use it to make deletions from an existing hypothesis clause with the same head. If this is not possible, then we use the counterexample to add a new clause to the hypothesis. We find any necessary additions when we add a new clause.

If $\mathbf{x}^h \cap \text{body}(c)^h$ is a negative instance, which we can determine by a membership query, then we can create a new smaller hypothesis clause whose body is $\mathbf{x} \cap \text{body}(c)$. (Notice that $\mathbf{x} \cap \text{body}(c) \subset \text{body}(c)$ because as a negative *counterexample*, $\mathbf{x}$ must satisfy c.)

To use $\mathbf{x}$ to add a new clause, we then use an idea from the revision algorithm for monotone DNF [9]. For each initial theory clause with the same head as we have associated (which for $\mathbf{F}$ is *all* initial theory clauses, since deletions of heads are allowed), use binary search from **x** intersect (the initial clause with the other heads set to 1) up to **x**. If we get to something negative with fewer than e additions, we update **x** to this negative example.

Whether or not **x** is updated, we keep going, trying all initial theory clauses with the associated head. This guarantees that in particular we try the initial theory clause with smallest revision distance to the target clause that **x** falsifies. All necessary additions to this clause are found by the calls to BinarySearch; later only deletions will be needed.

Algorithm 2 HornReviseUpToE(φ, e). Revises depth-1 acyclic Horn Sentence φ if possible using $\leq e$ revisions; otherwise returns failure.

```
 1: $H$ := everywhere-true empty conjunction
 2: while $(\mathbf{x} := \mathrm{EQ}(H)) \neq$ "Correct!" and $e > 0$ do
 3:   $h$ :=Associate($\mathbf{x}, \varphi$)
 4:   for all clauses $c \in H$ with head $h$ do
 5:     if $\mathrm{MQ}(\mathbf{x}^h \cap \mathrm{body}(c)^h) == 0$ then {delete vars from $c$}
 6:       $\mathrm{body}(c) = \mathrm{body}(c) \cap \mathbf{x}$
 7:       $e = e-$number of variables removed
 8:   if no vars. were deleted from any clause then {find new clause to add}
 9:     $min = e$
10:     $FoundAClause=false$
11:     for all $c \in \varphi$ with head $h$ (or all $c \in \varphi$ if $h == \mathbf{F}$) do
12:       $\mathbf{new} = \mathrm{body}(c)^h \cap \mathbf{x}^h$
13:       $numAddedLits = 0$ {# additions for this $c$}
14:       while $\mathrm{MQ}(\mathbf{new}) == 1$ and $numAddedLits < e$ do
15:         $l$ := BinarySearch($\mathbf{new}, \mathbf{x}^h$)
16:         $\mathbf{new} := \mathbf{new} \cup \{l\}$
17:         $numAddedLits = numAddedLits + 1$
18:         if $\mathrm{MQ}(\mathbf{x} - \{l\}) == 0$ then {$(\mathbf{x} - \{l\})$ is "Pivot"}
19:           {i.e., $\mathbf{x} - \{l\}$ is counterexample falsifying fewer target clauses}
20:           $\mathbf{x} = \mathbf{x} - \{l\}$
21:           restart the for all $c$ loop with this $\mathbf{x}$ by backing up to Line 11 to reset
              other parameters
22:       if $\mathrm{MQ}(\mathbf{new}) == 0$ then
23:         $\mathbf{x} := \mathbf{new}$
24:         $FoundAClause = true$
25:         $min := \min(numAddedLits, min)$
26:     if not $FoundAClause$ then
27:       return "Failure"
28:     else
29:       $H := H \wedge (x \rightarrow h)$ {treating $x$ as monotone disjunction}
30:       $e := e - min$
31: if $x ==$ "Correct!" then
32:   return $H$
33: return "Failure"
```

Theorem 1. *There is a revision algorithm for depth-1 acyclic Horn sentences with query complexity $O(d \cdot m^3 \cdot \log n)$, where d is the revision distance and m is the number of clauses in the initial formula.*

Proof sketch: We give here some of the highlights of the proof of correctness and query complexity of the algorithm; space does not permit detailed proofs. Relatively straightforward calculation shows that the query complexity of HornReviseUpToE for revising an initial formula of m clauses on a universe of n variables is polynomial in m, $\log n$, and the parameter e. Thus, if we can argue that when e is at least the revision distance the algorithm succeeds in finding the target, we are done.

The result follows from a series of lemmas. The first two lemmas give qualitative information. The first shows that the hypothesis is always one-sided (i.e., only negative counterexamples can ever be received), and the second says that newly added hypothesis clauses are not redundant.

Lemma 1. *The algorithm maintains the invariant that its hypothesis is true for every instance that satisfies the target function.*

Proof. Formally the proof is by induction on number of changes to the hypothesis. The base case is true, because the initial hypothesis is everywhere true.

For the inductive step, consider how we update the hypothesis, either by adding a new clause or deleting variables from the body of an existing clause.

Before creating or updating a clause to have head h and body $\mathbf{y}$, we have ensured that $\text{MQ}(\mathbf{y}^h) = 0$, that is, that $\mathbf{y}^h$ is a negative instance. Because of that, $\mathbf{y}^h$ must falsify some clause, and because of its form and the syntactic form of the target, it must be a clause with head h. None of the heads in $\mathbf{y}^h \setminus \mathbf{y}$ can be in any body, so $\mathbf{y}$ must indeed be a superset of the variables of some target clause with head h, as claimed.

Lemma 2. *If negative counterexample* $\mathbf{x}$ *is used to add a new clause with head h to the hypothesis, then the body of the new clause does not cover any target clause body covered by any other hypothesis clause with head h.*

Proof. Recall that head h was associated with $\mathbf{x}$. If $\mathbf{x}$ falsified the same target clause as an existing hypothesis clause body, then $\mathbf{x}$ would be used to delete variables from that hypothesis clause body. Therefore $\mathbf{x}$ does not falsify the same target clause as any existing hypothesis clause with the same head, and the newly added hypothesis clause's body is a subset of $\mathbf{x}$.

The following two lemmas, whose proof will be given in the journal version of this paper, complete the proof.

Lemma 3. HornReviseUpToE(φ, e) *succeeds in finding the target Horn sentence ψ if it has revision distance at most e from initial formula φ.*

Lemma 4. *The query complexity of* HornReviseUpToE *is $O(m^3 \cdot e \cdot \log n)$, where the initial formula has m clauses and there are n variables in the universe.*

3.2 Definite Horn sentences with unique heads

We give here a revision algorithm for definite Horn sentences with unique heads. Since we are considering only definite Horn sentences, note that the head variables cannot be fixed to 0. We use the algorithm for revising Horn sentences in the deletions-only model presented in [10] as a subroutine. Its query complexity is $O(dm^3 + m^4)$, where d is the revision distance and m is the number of clauses in the initial formula.

This algorithm has a first phase that finds all the variables that need to be added to the initial formula. That partially revised formula is then passed as an initial formula to the known algorithm [10] for revising Horn sentences in the deletions-only model of revision.

To find all necessary additions to the body b of clause $c = (b \to h)$, we first construct the instance $\mathbf{x}_c$ as follows: in the positions of the heads of the initial formula, instance $\mathbf{x}_c$ has 1's, except for a 0 in position h. Furthermore, instance $\mathbf{x}_c$ has 1's in all the positions corresponding to a variable in the clause's body, and 0 in all positions not yet specified.

Next, the query $\mathrm{MQ}(\mathbf{x}_c)$ is asked. If $\mathrm{MQ}(\mathbf{x}_c) = 0$, then no variables need to be added to the body of c. If $\mathrm{MQ}(\mathbf{x}_c) = 1$, the necessary additions to the body of c are found by repeated use of BinarySearch. To begin the binary search, $\mathbf{x}_c$ is the known positive instance that must satisfy the target clause c_* derived from c, and the assignment with a 0 in position h and a 1 everywhere else is the known negative instance that must falsify c_*.

Each variable returned by BinarySearch is added to the body of the clause, and $\mathbf{x}_c$ is updated by setting the corresponding position to 1. The process ends when $\mathbf{x}_c$ becomes a negative instance.

Once the necessary additions to every clause in the initial theory are found, a Horn sentence needing only deletions has been produced, and the deletions-only algorithm from [10] can be used to complete the revisions.

Theorem 2. *There is a revision algorithm for definite Horn sentences with unique heads in the general model of revision with query complexity $O(m^4 + dm^3 + d \log n)$, where d is the revision distance from the initial formula to the target formula.*

Proof sketch. The key part is adding variables to one initial clause $c = (b \to h)$. Let c_* be the target clause derived from c.

Lemma 5. *Every variable added to a clause c must occur in the target clause c_* that is derived from c.*

(Proof of lemma omitted.)

This is enough to allow us to use the earlier algorithm for learning in the deletions-only model.

The query complexity for the part of the algorithm that finds necessary additions is at most the $O(\log n)$ per added variable, which contributes a factor of $O((\varphi, \psi) \cdot \log n)$. The deletions-only algorithm has complexity $(m^4 + dm^3)$, where d is the revision distance. Combining these two gives us $O(m^4 + dm^3 + d \log n)$. $\square$

4 Revising threshold functions

We present a threshold revision algorithm ReviseThreshold. The overall revision algorithm is given as Algorithm 3, using the procedures described in Algorithms 4 and 5. Algorithm ReviseThreshold has three main stages. First we identify all the variables that are irrelevant in φ but relevant in ψ (Algorithm FindAdditions). Then we identify all the variables that are relevant in φ but irrelevant in ψ (Algorithm FindDeletions). Finally, we determine the target threshold. (In our pseudocode this third step is built into Algorithm FindDeletions, as the last iteration after the set of relevant variables of the target function is identified.)

Algorithm 3 The procedure ReviseThreshold(φ)

```
1: {function to be revised is $\varphi = \mathrm{TH}_U^t$}
2: Use 2 EQ's to determine if target is constant 0 or 1; if so **return**
3: $V :=$ FindAdditions($U$)
4: **return** FindDeletions($U, V$)
```

The main result of the section is the following.

Theorem 3. ReviseThreshold *is a threshold function revision algorithm of query complexity $O(d \log n)$, where d is the revision distance between the initial formula φ and the target function ψ.*

Proof sketch. Throughout, let $\varphi = TH_U^t$ and $\psi = TH_R^\theta$.

Algorithm 4 The procedure FindAdditions(U)

```
Require: the target function is not constant
1: $Potentials := X_n \setminus U$; $NewRelevants := \emptyset$
2: if MQ($\chi_U$) == 0 then
3:     $Base := U$
4: else
5:     $(Base, x) :=$ BinarySearch($\emptyset, U$), $Base := Base \setminus \{x\}$
6:     if MQ($\chi_{Base \cup Potentials}$) == 0 then
7:         return $\emptyset$
8: repeat
9:     $(Y, y) :=$ BinarySearch($Base, Base \cup Potentials$)
10:    $NewRelevants := NewRelevants \cup \{y\}$
11:    $Potentials := Potentials \setminus \{y\}$
12:    if MQ($\chi_{Base \cup Potentials}$) == 0 then
13:        $Base := Base \cup \{y\}$
14: until MQ($\chi_{Base}$) == 1
15: return $NewRelevants$
```

A set S is *critical for the target function* ψ, or simply *critical*, if $|S \cap R| = \theta - 1$. It is clear from the definition that if S is critical, then for every $Z \subseteq X_n \setminus S$ it holds that Z contains at least one relevant variable of ψ iff $\mathrm{MQ}(\chi_{S \cup Z}) = 1$.

Algorithm 5 The procedure FINDDELETIONS(U, V)

```
Require: $U$, $V$ disjoint; $V \subseteq R$ and $R \subseteq U \cup V$ ($R$ = relevant variables in target)
1: $\hat{U} := U$;
2: $u := |\hat{U}| + |V|$; $\ell := 1$
3: if ($\varphi' :=$ TESTEXTREME$(N, P, \hat{U} \cup V)) \neq$ constant then
4:     return $\varphi'$
5: while $u > \ell + 1$ do
6:     $m := \lceil (u + \ell)/2 \rceil$
7:     if ($\mathbf{x} := \mathrm{EQ}(\mathrm{TH}^m_{\hat{U} \cup V})) == YES$ then
8:         return $\mathrm{TH}^m_{\hat{U} \cup V}$
9:     let $C := \mathbf{x} \cap (\hat{U} \cup V)$
10:    if $\mathbf{x}$ is a positive counterexample then
11:        $P := C$ and $u := m$
12:    else
13:        $N := C$ and $\ell := m$
14: $(P, p) :=$ BINARYSEARCH$(\emptyset, P)$
15: $Base := P \cap N$, $P' := P \setminus Base$, $N' := N \setminus Base$
    {Now the key property holds for $Base$, $N'$ and $P'$}
16: $Test := (Base \cup P') \setminus \{p\}$
    {For any $i \in N'$, $\mathrm{MQ}(\chi_{Test \cup \{i\}}) = 1$ iff $i$ is relevant}
17: while $|P'| > 1$ do
18:    $i :=$ MAKEEVEN$(N', P', Base)$
       {Make $|N'|$ and $|P'|$ be even without spoiling the key property}
19:    if $\mathrm{MQ}(\chi_{Test \cup i}) == 0$ then
20:        $\hat{U} := \hat{U} - i$ and goto Line 2
21:    Let $N_0, N_1$ (resp. $P_0, P_1$) be an equal-sized partition of $N'$ (resp. $P'$)
22:    Ask $\mathrm{MQ}(\chi_{Base \cup N_j \cup P_k})$ for $j, k = 0, 1$
23:    Let $j$ and $k$ be indices s.t. $\mathrm{MQ}(\chi_{Base \cup N_j \cup P_k}) = 0$ {such $j$ and $k$ exist}
24:    $Base := Base \cup P_k$, $P' := P_{1-k}$, $N' := N_i$
25: $\hat{U} := \hat{U} \setminus N'$
26: goto Line 2
```

Procedure FINDADDITIONS (Algorithm 4) finds the new relevant variables; that is, the elements of $R \cap \bar{U}$, where $\bar{U} = X_n \setminus U$. It stores the uncertain but potentially relevant variables in the set *Potentials* (thus *Potentials* is initially set to $X_n \setminus U$). The procedure first determines a set $Base \subseteq U$ such that $Base$ is negative, and $Base \cup Potentials$ is positive (unless *Potentials* contains no relevant variables—in which case there are no new relevant variables used by ψ, so we quit). Then the search for the new relevant variables starts. We use BINARYSEARCH(*Base*, *Base* $\cup$ *Potentials*) to find one relevant variable. This variable is removed from *Potentials*, and the process is repeated, again using BINARYSEARCH. After removing a certain number of relevant variables from

Potentials, the instance $Base \cup Potentials$ must become critical. After reaching this point, we do not simply remove any newly found relevant variables from *Potentials*, but we also add them to the set *Base*. This way from then on it holds that $|(Base \cup Potentials) \cap R| = \theta$. Thus the indicator that the last relevant variable has been removed from *Potentials* is that *Base* becomes positive ($\mathrm{MQ}(\chi_{Base}) = 1$).

Let us assume that we have identified a set of variables that is known to contain all the relevant variables of the target function, and possibly some additional irrelevant variables. Consider threshold functions with the set of variables above, and all possible values of the threshold. Let us perform a sequence of equivalence queries with these functions, doing a binary search over the threshold value (moving down, resp. up, if a positive, resp. negative, counterexample is received). In algorithm FINDDELETIONS this is done by the first **while** loop starting at Line 5 and the use of TESTEXTREME right before it at Line 3 (the latter is needed to ensure the full range of search; it performs the test for the two extreme cases in the binary search, which the while loop might miss: the conjunction and the disjunction of all the variables). In case the relevant variables of the target functions are exactly those that we currently use, then one can see that this binary search will find ψ. Otherwise (i. e. when some of the currently used variables are irrelevant in ψ) it can be shown that after the above binary search we always end up with a "large" negative example (χ_N) and a "small" positive example (χ_P); more precisely they satisfy $|P| \leq |N|$. Using these sets one easily obtains three sets $Base, N'$ and P' that have the *key property*:

Key property: *Sets Base, N', and P' satisfy the key property if they are pairwise disjoint, and it holds that $Base \cup N'$ is negative, $|(Base \cup P') \cap R| = \theta$, and $|N'| \geq |P'|$.*

The following claim gives two important features of this definition.

Claim 1 a) *If $Base, N'$ and P' satisfy the key property then N' contains an irrelevant variable and P' contains a relevant variable.*

b) *If $Base, N'$ and P' satisfy the key property and $|P'| = 1$ then every element of N' is irrelevant.*

From now on we maintain these three sets in such a manner that they preserve the key property, but in each iteration the size of N' and P' get halved. For this we split up N' (respectively P') into two equal sized disjoint subsets N_1 and N_2 (resp. P_1 and P_2). When both $|N'|$ and $|P'|$ are even then we can do this without any problem; otherwise we have to make some adjustments to N' and/or to P', that will be taken care of by procedure MAKEEVEN (the technical details are omitted due to space limitations). Using the notation $\theta' = \theta - |R \cap Base|$ we have $|R \cap (N_1 \cup N_2)| < \theta'$ and $|R \cap (P_1 \cup P_2)| = \theta'$. Thus for some $j, k \in \{0, 1\}$ we have $|R \cap (N_j \cup P_k)| < \theta'$ (equivalently $\mathrm{MQ}(\chi_{Base \cup N_j \cup P_k}) = 0$). Note that the sets $Base := Base \cup P_k$, $N' := N_j$ and $P' := P_{1-k}$ still have the key property, but the size of N' and P' is reduced by half. Thus after at most $\log n$ steps P' is reduced to a set consisting of a single (relevant) variable. Thus N' is a nonempty set of irrelevant variables (part *b)* of Claim 1). $\square$

We mention some additional results showing that both types of queries are necessary for efficient revision, the query bound of algorithm ReviseThreshold cannot be improved in general and that Winnow (see [19]) cannot be used for revising threshold functions (at least in its original form).

Theorem 4. a) *Efficient revision is not possible using only membership queries, or only equivalence queries.*

b) *The query complexity of any revision algorithm for threshold functions is* $\Omega\left(d \log \frac{n}{d}\right)$ *queries, where* d *is the revision distance.*

c) *Winnow is not an efficient revision algorithm for threshold functions because it may make too many mistakes. More precisely, for any weight vector representing the initial threshold function* $\mathrm{TH}^1_{x_1,\ldots,x_n}$*, Winnow can make* n *mistakes when the target function is* $\mathrm{TH}^2_{x_1,\ldots,x_n}$.

Proofs will be given in the full version of this paper.

References

1. Blum, A., Hellerstein, L., Littlestone, N.: Learning in the presence of finitely or infinitely many irrelevant attributes. J. of Comput. Syst. Sci. **50** (1995) 32–40 Earlier version in 4th COLT, 1991.
2. Bshouty, N., Hellerstein, L.: Attribute-efficient learning in query and mistake-bound models. J. of Comput. Syst. Sci. **56** (1998) 310–319
3. Koppel, M., Feldman, R., Segre, A.M.: Bias-driven revision of logical domain theories. Journal of Artificial Intelligence Research **1** (1994) 159–208
4. Ourston, D., Mooney, R.J.: Theory refinement combining analytical and empirical methods. Artificial Intelligence **66** (1994) 273–309
5. Richards, B.L., Mooney, R.J.: Automated refinement of first-order Horn-clause domain theories. Machine Learning **19** (1995) 95–131
6. Towell, G.G., Shavlik, J.W.: Extracting refined rules from knowledge-based neural networks. Machine Learning **13** (1993) 71–101
7. Wrobel, S.: Concept Formation and Knowledge Revision. Kluwer (1994)
8. Wrobel, S.: First order theory refinement. In De Raedt, L., ed.: Advances in ILP. IOS Press, Amsterdam (1995) 14–33
9. Goldsmith, J., Sloan, R.H., Turán, G.: Theory revision with queries: DNF formulas. Machine Learning **47** (2002) 257–295
10. Goldsmith, J., Sloan, R.H., Szörényi, B., Turán, G.: Theory revision with queries: Horn, read-once, and parity formulas. Artificial Intelligence **156** (2004) 139–176
11. Mooney, R.J.: A preliminary PAC analysis of theory revision. In Petsche, T., ed.: Computational Learning Theory and Natural Learning Systems. Volume III: Selecting Good Models. MIT Press (1995) 43–53
12. Sloan, R.H., Szörényi, B., Turán, G.: 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, USA, August 24-27, 2003, Proceedings. Volume 2777 of Lecture Notes in Artificial Intelligence., Springer (2003) 625–639
13. Doshi, J.U.: Revising Horn formulas. Master's thesis, Dept. of Computer Science, University of Kentucky (2003)

14. Angluin, D., Frazier, M., Pitt, L.: Learning conjunctions of Horn clauses. Machine Learning **9** (1992) 147–164
15. Valiant, L.G.: Projection learning. Machine Learning **37** (1999) 115–130
16. Pinker, S.: The Blank Slate: The Modern Denial of Human Nature. Viking Press (2002)
17. Valiant, L.G.: A neuroidal architecture for cognitive computation. Journal of the ACM **47** (2000) 854–882
18. Valiant, L.G.: Robust logics. Artificial Intelligence **117** (2000) 231–253
19. Littlestone, N.: Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. Machine Learning **2** (1988) 285–318
20. Uehara, R., Tsuchida, K., Wegener, I.: Optimal attribute-efficient learning of disjunction, parity, and threshold functions. In: Computational Learning Theory, Third European Conference, EuroCOLT '97, Jerusalem, Israel, March 1997, Proceedings. Volume 1208 of Lecture Notes in Artificial Intelligence., Springer (1997) 171–184
21. Hegedűs, T., Indyk, P.: On learning disjunctions of zero-one threshold functions with queries. In: Algorithmic Learning Theory, 8th International Workshop, ALT '97, Sendai, Japan, October 1997, Proceedings. Volume 1316 of Lecture Notes in Artificial Intelligence., Springer (1997) 446–460
22. Hegedüs, T.: On training simple neural networks and small-weight neurons. In: Computational Learning Theory: EuroColt '93. Volume New Series Number 53 of The Institute of Mathematics and its Applications Conference Series., Oxford, Oxford University Press (1994) 69–82
23. Pitt, L., Valiant, L.G.: Computational limitations on learning from examples. J. ACM **35** (1988) 965–984
24. Schmitt, M.: On methods to keep learning away from intractability. In: Proc. International Conference on Artifical Neural Networks (ICANN) '95. Volume 1. (1995) 211–216
25. Sloan, R.H., Turán, G.: Learning from incomplete boundary queries using split graphs and hypergraphs. In: Computational Learning Theory, Third European Conference, EuroCOLT '97, Jerusalem, Israel, March 1997, Proceedings. Number 1208 in Lecture Notes in Artificial Intelligence, Springer (1997) 38–50
26. Angluin, D.: Learning propositional Horn sentences with hints. Technical Report YALEU/DCS/RR-590, Department of Computer Science, Yale University (1987)
27. Hammer, P.L., Kogan, A.: Quasi-acyclic propositional Horn knowledge bases: optimal compression. IEEE Trans. Knowl. Data Eng. **7** (1995) 751–762
28. Arimura, H.: Learning acyclic first-order Horn sentences from entailment. In: Algorithmic Learning Theory, 8th International Workshop, ALT '97, Sendai, Japan, October 1997, Proceedings. Volume 1316 of Lecture Notes in Artificial Intelligence., Springer (1997) 432–445
29. Angluin, D.: Queries and concept learning. Machine Learning **2** (1988) 319–342