More Theory Revision with Queries

[Extended Abstract]

Judy Goldsmith*
Dept. of Computer Science
University of Kentucky
763 Anderson Hall
Lexington, KY 40506-0046
goldsmit@cs.uky.edu

Robert H. Sloan†
Dept. of EE & Computer Science
University of Illinois at Chicago
851 S. Morgan St. Rm 1120
Chicago, IL 60607-7053
sloan@eecs.uic.edu

ABSTRACT

Given a Boolean formula that is not quite right, how does one fix it? That is, if a given formula differs from an unknown target formula, what is the complexity of revising the given formula? The tools available for determining the revisions are queries to membership and equivalence oracles, namely, questions of the form: "Is this an instance of the target formula," and "Is this hypothesis equivalent to the target formula?" In the latter case, if the answer is "No," the oracle returns an instance that is true for exactly one of the hypothesis and the target. For Horn sentences that require only deletion revisions, a revision algorithm is given that is polynomial in the number of clauses of the formula and the minimum number of deletions needed. For 2-term monotone DNF formulas, a revision algorithm is given that is polynomial in the minimum number of necessary deletions and additions and the logarithm of the number of variables. Previous work addressed deletion-only revisions to 2-term unate DNF formulas.

1. INTRODUCTION

Consider the following situation. You hire a domain expert (whom we shall call "Mommy") and a knowledge engineer to develop an expert system for predicting what your picky preschooler will eat. The following "pretty close" initial theory is developed:

WillEat := *VeryBland* OR ((NOT *vegetables*) AND *bland* AND *Meat*).

Then, though you use the initial theory as a general guide, you happen to observe the preschooler consume a full pound of grilled lamb ribs in spicy black pepper-teriyaki marinade. You must *revise* or *edit* your initial theory, perhaps to either

WillEat := *VeryBland* OR ((NOT *vegetables*) AND *Meat*),

or

WillEat := *VeryBland* OR ((NOT *Vegetables*) AND *bland* AND *Meat*) OR *Lamb*.[1]

This is the problem known in machine learning as *theory revision* (or *knowledge-base revision*) (e.g., [10, 15, 16, 19]). There is a very large body of work on learning from examples, that is, revising the empty theory to the correct theory; theory revision is of independent interest to the greater Computer Science community because it arises whenever databases (and particularly, large databases) must be revised. Unfortunately, most very large theories based on real-world examples are not exactly correct. Our work shows that, at least for certain forms of theories, a few corrections to the theory can be made at a much lower cost in terms of examples (i.e., training data, or queries in formal models) than (re-)learning the whole theory from scratch.

Sloan and Turàn [18] provided a formal model for theory revision in the computational learning theory framework, and some initial results. Algorithms are permitted resources polynomial in the number of required revisions, but only polylogarithmic in the total number of variables and the size of the initial theory. That paper gave two models for permitted revisions, both based on models used earlier in the machine learning and other related literature. The more limited model, which we will call the *deletions-only* model, permits revisions of a propositional Boolean formula by fixing any occurrence of any variable to either 1 or 0. The broader model, which we will call the *general* model, permits the addition of new variables to clauses or terms of the formula, as well as permitting variables to be fixed to constants.

Sloan and Turàn provided an algorithm in the deletions-only model for both 2-term unate DNF and read-once formulas. In the general model, a revision algorithm is given for threshold functions and the parity function.

*Partially supported by NSF grant CCR-9610348; work done while visiting the Dept. of EECS at the University of Illinois at Chicago.

†Partially supported by NSF grant CCR-9800070.

[1]This theory seems to work well for the second author's five-year old.

In this paper we extend the positive result for unate 2-term DNF in two significant directions. First, in the deletions-only model, we present an algorithm for revising $O(\log n)$-clause Horn sentences (i.e., conjunctions of Horn clauses). Second, we present an algorithm for revising monotone 2-term DNF in the general model. The first result is important because, while monotone DNF is a somewhat limited syntactic form, Horn sentences are rich enough that they are a common choice for practical machine learning systems (e.g., Koppel's system [10], and almost all systems that learn logic programs).

1.1 More details about the model

A common assumption in the AI theory revision literature (e.g., [10]) is that one is given a theory together with a set of labeled examples, and the set always includes some that disagree with the classification provided by the theory. Typically one is supposed to make "small" revisions to the theory so that it classifies all the given examples correctly, though small is often left undefined (and finding the "smallest" revision is usually NP-complete).

Sloan and Turán [18] formulated the theory revision problem in the standard model of membership and equivalence queries [1], and we will use their model here as well. That is, there is one oracle MQ($\cdot$) that tells whether any given instance is positive or negative according to the correct theory, and a second oracle EQ($\cdot$) that responds to a false conjecture about the correct theory with a counterexample. In an expert system setting, the expert would function as the membership oracle. The problem of revising Horn sentences in the general model (which we do not solve here) *without* using membership queries is at least as hard as the well-known open problem of PAC learning DNF formulas. An algorithm for revising Horn sentences in a query model could be used to learn Horn sentences in that model by revising the empty initial theory. One can easily convert an algorithm for learning from equivalence queries to one for PAC learning; reductions in Kearns et al. [8] show that PAC-learnability of Horn sentences would imply PAC-learnability of general DNF or CNF formulas.

The formal concept class for revisions of a Horn sentence in the deletions-only model consists only of formulas obtained from the φ_0, the initial formula to be revised, by fixing one or more of its variables to 0 or 1. The equivalence queries used in our learning algorithm for Horn sentences are not necessarily in the formal concept class to be learned. They are always Horn sentences, but the correspondence between initial clauses and query clauses is not always one to one. However, the final output of our algorithm *is* a formula that can be obtained by fixing variables of φ_0.

1.2 Related work

In machine learning, there is a large body of work sometimes called *theory refinement* (see, e.g., Wrobel [22, 23]). Theory refinement includes *theory restructuring*, which is aimed at making a theory more efficient or transparent, as well as theory revision. "Theories" are also referred to as *knowledge-bases*, or in computational learning theory as *concepts*. There are numerous systems for theory revision in propositional and predicate logic (e.g., [10, 15, 16, 19]). Such systems usually have not come with any formal guarantees of performance; rather they are evaluated empirically. Some formal results were obtained by Koppel et al. [3, 10].

Probably the three previous papers that are closest to this one are by Mooney [14], Greiner [7], and Sloan and Turán [18]. We discuss each one in turn.

Mooney [14] formulated an approach to the study of theory revision in computational learning theory based on *syntactic distances*. The syntactic distance between a given concept representation and another concept is the minimal number of elementary operations (such as the addition or the deletion of a literal or a clause) needed to transform the given concept representation to a representation of the other concept. Mooney proposed considering the PAC-learnability of the class of concepts having a bounded syntactic distance from a given concept representation, and gave a positive result for *sample (or query) complexity*, but did not consider computational complexity.

Greiner's work [7] is in the same spirit as Mooney's, and gives mostly positive results for sample complexity, and mostly negative (i.e., hardness) results for polynomial time PAC learnability. Greiner has results concerning both predicate and propositional logic, but since our results in this paper concern only propositional logic, we will discuss only Greiner's results on propositional logic. Greiner's results give further evidence that membership queries may be necessary for revising Horn sentences. He shows that a decision problem involving revising Horn sentences to be consistent with a sample is NP-hard. Since learnability from equivalence queries alone implies that one can solve such consistency problems, Greiner's results imply a hardness result for the query learning model as well. However, Greiner's model differs from what we consider in this paper, in that his instances are Horn clauses (which either are or are not entailed by a Horn sentence theory), whereas our instances are assignments to all propositional variables (which, for our results on Horn sentence theories, either do or do not contradict the Horn sentence).

Sloan and Turán [18] formulated the theory revision problem in a manner similar to Mooney, using syntactic distances in a query model instead of the PAC model. They required query complexity and computational complexity polynomial in the number of revisions and polylogarithmic in the total number of variables. (This is similar to the notion of efficiency for *attribute efficient* learning algorithms [4, 5].) Two different measures of syntactic distance were considered, the deletions-only and general models mentioned previously. They gave positive results for 2-term unate DNF, unate k-DNF, and read-once formulas in the deletions model. Also, a hardness result was given, showing that it is impossible to revise a monotone DNF formula on n variables with $O(n)$ terms needing a single revision without using $\Omega(n)$ queries. Positive results were given in the general model for threshold and parity functions.

The problem of correcting errors is pervasive, and problems somewhat related to theory revision show up in several places. Among them are fault analysis of circuits in switching theory (see, e.g., Kohavi [9]), program debugging (e.g., [17]), and model-based diagnosis (see, e.g., [6, 11, 13]). See Sloan and Turán [18] for a somewhat longer discussion of these connections.

1.3 Unrelated work

One can consider revising a database, rather than a knowledge base. In logic programming, this can be modeled as a *revision logic program,* which specifies necessary changes to

a database using atoms IN(A) and OUT(A). The underlying computational question is, given one database and a revision logic program, to revise the database so that all revisions are justified based on the program [12].

Another problem with a similar name is the *minimum edit* or *revision distance* problem. That problem is one of calculating the minimum number of operations needed to transform one given, fixed string, into another, and it has long had a well known dynamic programming algorithm, usually attributed to Wagner and Fischer [21].

2. DEFINITIONS AND NOTATION

We use standard notions from propositional logic such as variable, term, disjunctive normal form (DNF), monotone, clause, cover, etc. We assume throughout that the concepts TRUE and FALSE are allowed as equivalence queries.

A Horn clause has 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 no unnegated variables will be considered to have head $\mathbf{F}$; a clause with no negated variables (a *fact*) to have body $\mathbf{T}$.

When convenient, we treat both Horn clause bodies and monotone DNF terms as vectors in $\{0,1\}^n$, and vectors sometimes as subsets of $[n]$. If for $x \in \{0,1\}^n$ and clause C we have body(C) $\subseteq x$, we say x *covers* C. Notice that x *falsifies* C iff x covers C and 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.

Let φ be a Boolean formula using the variables $x_1, \ldots, x_n$. Then $C_\varphi \subseteq \{0,1\}^n$ is the concept represented by φ, that is, C_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 C_φ is $\{110, 101, 111\}$. 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 $\mathcal{C}_\varphi$. 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 $x_1 \wedge x_2$. Notice that for Horn sentences this revision operator could alternately be viewed as three operators: *deleting one variable from a body* (by fixing the occurrence of this variable to true), *deleting a clause* (by fixing its head to true or one of its body variables to false), and *changing the head of a clause to* $\mathbf{F}$.

The *revision distance* between a formula φ and some other concept C 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. In the deletions-only model, our revision operator is *fixing an occurrence of a variable* in a formula to a constant. In the general model, adding a new variable to φ is also a revision operator. Thus, for example, 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 in either model.

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 number of variables.

Let h be a collection of Horn clauses. Define the $\dot{\cap}$ operation with respect to h (which will, in Section 3, be understood to be the current constructed hypothesis) as follows: The result of $x \dot{\cap} y$ is the same as the result of $x \cap y$ *except* when there is a hypothesis clause C such that $x \cap y$ covers body(C) and x has a 1 in the position head(C), in which case that 1 stays on regardless of y.

We always use $\subset$ to denote *strict* subset.

3. REVISING PROPOSITIONAL HORN SENTENCES

We present Algorithm REVISEHORN for revising Horn sentences in the deletions-only model. It uses some general ideas from the algorithm of Angluin et al. [2] for learning Horn sentences from membership and equivalence queries. Indeed, the top-level REVISEHORN is essentially identical to Angluin et al.'s algorithm. However, the details of the subroutines, especially NEWMETACLAUSE, are different—this is where we use the the initial theory φ_0 to make the query complexity independent of the number of Boolean variables, and dependent on only the revision distance between φ_0 and the target.

Following Angluin et al., we find it convenient to organize our hypothesis by distinct clause *bodies*. We call the collection of all clauses that have the same body a *meta-clause*.

Algorithm 1 REVISEHORN. Revises Horn sentence φ_0.

```
1:  h = empty hypothesis (everywhere true)
2:  while (x = EQ(h)) ≠ "Correct" do
3:    if h(x) == 1 then {x is a negative counterexample}
4:      for each meta-clause C ∈ h in order do
5:        if body(C) ∩ x ⊂ body(C) and then
          MQ(body(C) ∩ x) == 0 then
6:          Replace C with SHRINKBODY(C, body(C) ∩
            x, φ0, h) and break the for loop
7:      h = h∧ NEWMETACLAUSE(x, φ0, h)
8:    else {x is a positive counterexample}
9:      for each meta-clause C such that C(x) = 0 for at
        least one head of C do
10:       FIXHEADS(x, C, h)
```

The basic idea of REVISEHORN is as follows. We start with the empty conjunction (i.e., everything is classified as true) and repeatedly make equivalence queries until we are done. If possible, negative counterexamples are used to edit the body of a meta-clause in the current hypothesis. If this is not possible for a particular negative counterexample, then it is used to add a new meta-clause to the hypothesis. (That case will always occur with the first counter example.) Positive counterexamples are always used to edit the heads of existing meta-clauses.

We can use certain negative counterexamples to create a new meta-clause, because every negative instance falsifies some target clause. Notice that if negative counter example x falsifies target clause C_* that is a revision of some initial theory clause C_0, then $x \cap$ body(C_0) also falsifies C_*. Thus, for each clause C_0 of the initial theory, we would like to say that if MQ($x \cap$ body(C_0)) $= 0$, then set x to $x \cap$ body(C_0).

However, there are two issues to which we must pay careful attention.
First, we do not want to rediscover any meta-clauses already in our hypothesis. Perhaps x falsified only target clauses not falsified by any meta-clause body in our hypothesis, but $x \cap \text{body}(C_0)$ falsifies some target clause that is falsified by a meta-clause body already in the current hypothesis. This is checked by NEWMETACLAUSE in Line 5.
Second, 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 $x \stackrel{\bullet}{\cap} C_0$ instead of $x \cap C_0$.

Algorithm 2 NEWMETACLAUSE(x, φ_0, h). Intersects negative example x that generates a new hypothesis meta-clause with all initial theory clauses.

1: **for** each clause C of the initial theory φ_0 **do**
2: $b = x \stackrel{\bullet}{\cap} C$
3: **if** MQ$(b) == 0$ **then** $\{x \stackrel{\bullet}{\cap} C$ is "new" negative counterexample$\}$
4: **for** each meta-clause M of the hypothesis h **do**
5: **if** $b \cap \text{body}(M) \subset \text{body}(M)$ **and then** MQ$(b \cap \text{body}(M)) == 0$ **then**
6: **exit** and SHRINKBODY$(M, \text{b}, \varphi_0, h)$
7: **return** new meta-clause with body x and head **F**

Next we describe how to handle positive counterexamples to our hypothesis. There are two cases: The counterexample x falsifies a hypothesis clause of the form $(foo \rightarrow \mathbf{F})$ and x falsifies a hypothesis clause with a head. In the first case, we add all possible heads. In the second case, we delete the extra head.
Specifically, FIXHEADS$(x, (foo \rightarrow \mathbf{F}), \varphi_0)$ adds as heads of the meta-clause body *foo* all heads that occur in clauses of φ_0 and do not occur in *foo*.

Algorithm 3 SHRINKBODY(C, x, φ_0, h)

Require: x such that $x \cap \text{body}(C) \subset \text{body}(C)$ and $h(x) = 1$
for Each clause $C_0 \in \varphi_0$ **do**
$b = x \stackrel{\bullet}{\cap} \text{body}(C_0)$
if MQ$(b) = 0$ **then**
$x = b$
body$(C) = x \cap \text{body}(C)$
if head$(C) \neq \mathbf{F}$ **then**
Add to head(C) any variable just deleted from body(C) that is the head of some clause of φ_0

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 hypothesis meta-clauses are created only by calls to NEWMETACLAUSE and meta-clause bodies are altered only by calls to SHRINKBODY. The set of heads of a meta-clause can be altered only by calls to FIXHEADS and to SHRINKBODY.

PROPOSITION 1. *If the current hypothesis h of Algorithm* REVISEHORN *classifies x as positive, that is, $h(x) = 1$, then for any y, we have $h(x \stackrel{\bullet}{\cap} y) = 1$.*

Proposition 1 is straightforward, and we implicitly use it in several of our later proofs.

LEMMA 2. *Each meta-clause body in the hypothesis always falsifies some clause of the target concept.*

PROOF. The body of the meta-clause is always a negative instance of the target. This is true when the meta-clause is first added by NEWMETACLAUSE, and this is maintained as an invariant because it is guaranteed by a membership query immediately before changing a meta-clause body in SHRINKBODY. □

LEMMA 3. *For every hypothesis meta-clause C with head other than **F**, for every target clause C_* that body(C) falsifies, head(C_*) is always one of the heads of C.*

PROOF. We put in all possible heads when we first change the meta-clause head from **F**. When we delete a variable from a meta-clause 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. □

LEMMA 4. *No two hypothesis meta-clauses ever falsify the same target clause.*

Proof sketch. We follow the proof in Angluin et al. [2] 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 hypothesis meta-clause bodies $b_1, b_2, \ldots, b_h$ 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 both C_k and C_ℓ falsify the C_*, and WLOG, $k < \ell$. This contradicts the claim, since C_k falsifies C_*.
Now we prove that the claim is true by induction on the number of changes made to the hypothesis. This 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 bodies, so we need consider only negative counterexamples.
Consider first the case of shrinking a meta-clause body b_j in a call to SHRINKBODY(C_j, x, φ_0, h), where $b_j = \text{body}(C_j)$, $C_j \in h$, and x is a negative counterexample for hypothesis h. After this change b_j can cover only fewer clauses of the target formula than before, so we need worry only about meta-clauses 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_i covered C_* and so x falsified C_*. Since b_i covers C_*, then $b_i \cap x$ falsifies C_*, and x would have been used to refine b_i, as long as $x \cap b_i \subset b_i$. What happens if $b_i = x \cap b_i$? Since $x \cap b_i$ falsifies C_*, we have that b_i falsifies C_*. By Lemma 3, b_i's meta-clause has either head(C_*) or **F** as its head. Therefore x does not satisfy b_i's meta-clause, contradicting the assumption that x is a *negative counterexample* (i.e., x satisfies all hypothesis clauses)[2].

[2]Note that it is crucial here that we include the heads of other clauses as they appear in x, in order to not introduce a spurious negative membership query. This is precisely why we introduced $\stackrel{\bullet}{\cap}$.

Next, consider adding a new meta-clause using negative counterexample x to get new meta-clause body b from NEWMETACLAUSE(x, h, φ_0). Suppose b falsifies C_*, and $b_i \in h$ covers C_*. Then in Line 7 of NEWMETACLAUSE we would exit and use b to edit b_i as long as $b \cap b_i \subset b_i$. Otherwise, $b \cap b_i = b_i$, so b_i falsifies C_*, and again by Lemma 3, it must be that b does not satisfy b's meta-clause, contradicting the assumption that $h(b) = 1$. □

THEOREM 5. *Algorithm* REVISEHORN *will revise a Horn sentence containing m clauses and needing e revisions using at most $O(m^3 e)$ queries.*

Proof sketch. First, observe that once a particular meta-clause is added, it is never deleted. This follows from Lemmas 2 and 3: Lemma 2 says that the meta-clause body always falsifies some target clause, and Lemma 3 says that therefore it will always have some head.

In the worst case, one meta-clause C is introduced into the hypothesis for each target clause C_*. Let us consider how many queries that one meta-clause C can generate over the lifetime of the algorithm. In its creation by NEWMETACLAUSE, C can generate $O(m^2)$ queries.

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

The meta-clause may have superfluous variables which must be edited out at the cost of $O(m)$ queries per edit, using SHRINKBODY. If the meta-clause is associated with a target clause C_* then it requires at most e such deletions. However, in the course of the algorithm these associations may shift as many as m times. Thus, there may be as many as em^2 queries associated with deletions from a given meta-clause. This is the dominant factor in the analysis.

Since there are up to m clauses, the total algorithm requires $O(em^3)$ queries. □

4. REVISING 2-TERM MONOTONE DNF FORMULAS

In this section we present an algorithm for revising monotone 2-term DNF in the general model of theory revision. This algorithm proceeds in stages. During Stage e, we run an $O(e \log n)$ time algorithm that revises the given hypothesis and finds the target formula if that is possible in e edits. The main work is done by Algorithm REVISEUPTOE, which is given an *a priori* bound e on the revision distance between the initial formula and the target. Before presenting that algorithm, we begin by describing some technical details needed in it.

Binary search

If $Y \subseteq Z$ and $\mathrm{MQ}(Y) = 0$ and $\mathrm{MQ}(Z) = 1$, we can cover a term in the target using Y and one or more variables from $Z - Y$. This situation might arise, for instance, if Y is one of the original terms of the target, and Z is the term containing all variables.

The basic idea is to repeatedly use binary search to find individual variables that *must* be added to cover a term in the target.

We present Algorithm BINARYSEARCH as Algorithm 4. It is given as input the maximum number of edits e that it may make, and returns the edited term and the number of edits remaining.

Algorithm 4 BINARYSEARCH(Y, Z, e). Finds necessary additions to Y from Z to cover a target term, if this can be done with $\leq e$ additions and if Z covers a unique extension of Y that covers a target term. If there are two such extensions, it sets a flag, *PivotFlag*, and returns a nonambiguous Z. Initially, $Y \subset Z$, and $\mathrm{MQ}(Y) = 0$ and $\mathrm{MQ}(Z) = 1$.

```
1: if e ≤ 0 then {All edits already used}
2:    return "Failure"
3: S = Y, T = Z.
4: while |T − S| > 1 do {binary search for 1 var.}
5:    Divide T − S into equal-size sets d1 and d2.
6:    if MQ(S ∪ d1) = 0 then
7:       S = S ∪ d1
8:    else
9:       T = S ∪ d1
10: Let v be the unique variable in T − S.
11: if MQ(Z − v) = 1 then {Z covered 2 target terms; v
    is in only one}
12:    if Caller already had a one term hypothesis then
13:       return "Failure"
14:    else {Occurs only with call from Line 27 of REVISE-
       UPTOE}
15:       return PivotFlag and Z − v
16: Y = Y ∪ v
17: if MQ(Y) = 1 then
18:    return Y and e − 1
19: else
20:    return BINARYSEARCH(Y, Z, e − 1)
```

The **while** loop in lines 4–9 does the actual binary search for one variable from $Z - Y$ that must be added to the term Y. (A similar idea was used by Uehara, Tsuchida, and Wegener [20].) Eventually, we find a single variable v that makes the difference between a positive and negative membership query.

If $\mathrm{MQ}(Y \cup v) = 1$, then we are done. Otherwise, we must iterate and look for another variable to add. We wrote this as a recursive call; it is in lines 16–20 of the pseudocode.

One special case arises: this search might add variables from both target terms. In particular, imagine that in the course of a binary search from Y to Z we found that the variable v needs to be added; that is, there is some set A of variables such that $\mathrm{MQ}(Y \cup A) = 0$ and $\mathrm{MQ}(Y \cup A \cup \{v\}) = 1$. In addition, suppose that $\mathrm{MQ}(Z - \{v\}) = 1$. Then it must be that $Z - \{v\}$ contains one term, and $Y \cup A \cup \{v\}$ contains the other. We call such a v a *pivot*, and we check for a pivot in lines 11–15 of the pseudocode.

If a pivot is found and the caller already has a one term hypothesis, then the caller cannot use two more terms. Therefore, in such a case, Binary Search must return "failure." This is tested for at line 12.

If the caller does not yet have a term in its hypothesis, we return a flag we call *PivotFlag* and a modified positive example that must satisfy only one term: the original positive instance with the pivot variable changed to be 0, namely, $Z - \{v\}$. The caller can use this to back up and start over without the ambiguity caused by having a pivot.

Except in the pivot case, the algorithm BINARYSEARCH(Y, Z, e) either fails, or returns e' and a set S such

that $Y \subset S \subseteq Z$ and $|S - Y| = e - e'$ and $\mathrm{MQ}(S) = 1$ and such that for the unique target term $G \subseteq S$, $S - Y \subseteq G$.

Ambiguity

The algorithm constructs a hypothesis; at each stage of the construction, there are 0,1, or 2 terms in the hypothesis. The initial formula's terms are T_1 and T_2. Let x be a positive counterexample to our current hypothesis. For some target term G, $G \subseteq x$. G can be obtained by editing T_1 or T_2, but it is not obvious which one to edit in order to achieve the minimum total number of edits. Thus, at certain points in the pseudocode for Algorithm ReviseUpToE, the words "Try both:" appear. In this case, we first try to finish learning based on our current hypothesis and the assumption that target term G should be obtained by editing T_1. If this succeeds, we are done. If this fails, or starts to use more than e total revisions, then we instead try the assumption that G should be obtained by editing T_2. This situation arises only when there is ambiguity about which target terms are associated with which initial terms. The algorithm constructs a hypothesis so that, if there are two hypothesis terms, each covers a distinct target term. Thus, the ambiguity will not arise with respect to the constructed hypothesis, and at most once with respect to the initial terms. Therefore, the branching implicit in "Try both:" affects our query complexity by only a multiplicative factor of 2.

RefineDown

If we have a hypothesis whose term(s) cover terms of the target, we are in the situation of revising monotone DNF in the deletions-only model. We use a subroutine Algorithm RefineDown, similar to the main subroutine used by Sloan and Turán for 2-term monotone DNF in the deletions-only model [18]. It updates the number of edits remaining, and, in the case of a one-term hypothesis, returns (success or failure or) an edit of that term and a positive counterexample.

Algorithm 5 Algorithm RefineDown(h, e). Input h is the current hypothesis, all of whose term(s) must cover a distinct term of the target hypothesis. Input e is the number of remaining edits allowed, and is updated as we go. If either the correct hypothesis is found or the error limit exceeded, the procedure halts immediately.

```
while (x = EQ(h)) ≠ "Correct" do
  for Each term t ∈ h do
    if MQ(x ∩ t) == 1 then
      e = e − |t \ t ∩ x|
      if e < 0 then
        return "Failure"
      t = t ∩ x
  if no term of h was revised by counterexample x then
    if h has two terms then
      return "Failure"
    else
      return h, x, e
```

We next make a few remarks about Algorithm ReviseUpToE, explaining the various cases that it has.

The Algorithm

There are two main cases, depending on whether there is any target term contained in $T_1 \cap T_2$, where the initial for-

Algorithm 6 ReviseUpToE$(T_1 \vee T_2, e)$. Revises $T_1 \vee T_2$ if possible using at most e revisions; otherwise returns "Failure." Note that if any subroutine either finds the correct hypothesis or returns "Failure," then this algorithm also terminates. Also, if the error limit e is ever exceeded, this algorithm terminates immediately and returns "Failure".

```
 1: if MQ(T1 ∩ T2) == 1 then
 2:   e0 = e
 3:   (h, x, e) = RefineDown(T1 ∩ T2, e)
 4:   if MQ(x ∩ T1 ∩ T2) == 1 then
 5:     RefineDown(h ∨ (x ∩ T1 ∩ T2), e)
 6:   else if MQ(x ∩ Ti) == 1 for exactly one i ∈ {1,2}
      then
 7:     Try both:
 8:     A. RefineDown(h ∨ Ti ∩ x, e0 − |Ti \ h| − |Ti \ (Ti ∩ x)|)
 9:     B. (S, e) = BinarySearch(x ∩ Tī, x, e0 − |Ti \ h|)
10:     RefineDown(h ∨ S, e)
11:   else {Both MQ(x ∩ Ti) are 0.}
12:     Try both with i = 1, 2:
13:     (S, e) = BinarySearch(x ∩ Ti, x, e0 − |Tī \ h|)
14:     RefineDown(h ∨ S, e)
15: else {No target term wholly inside intersection T1 ∩ T2}
16:   if MQ(T1) == MQ(T2) == 1 then
17:     RefineDown(T1 ∨ T2, e)
18:   else if MQ(Ti == 1) for exactly one i ∈ {1,2} then
19:     Try both:
20:     A. (h, x, e') = RefineDown(Ti, e)
21:     (S, e) = BinarySearch(x ∩ Tī, x, e')
22:     RefineDown(h ∨ S, e)
23:     B. (h, e) = BinarySearch(Tī ∩ Ti, Ti, e)
24:     (h, x, e) = RefineDown(h, e)
25:     (S, e) = BinarySearch(x ∩ Ti, x, e)
26:     RefineDown(h ∨ S, e)
27:   else {Both MQ(Ti) are 0}
28:     x = EQ(FALSE)
29:     Try both with i = 1, 2:
30:     if BinarySearch(x ∩ Ti, x, e) returns PivotFlag
        and z then
31:       Reset x to be z and go back to line 29
32:     else
33:       Let (S, e) be what BinarySearch returned
34:       (h, y, e) = RefineDown(S, e)
35:       (S', e) = BinarySearch(y ∩ Tī, y, e)
36:       RefineDown(h ∨ S', e)
```

mula is $T_1 \vee T_2$. There is a target term in $T_1 \cap T_2$ if and only if $\mathrm{MQ}(T_1 \cap T_2) = 1$. If there is, then our initial hypothesis will be the single term $T_1 \cap T_2$ and we may eventually delete variables from that term, add and edit a second term, or both ("Case 1"). If not ("Case 2"), then if $\mathrm{MQ}(T_1) = \mathrm{MQ}(T_2) = 1$, then our initial hypothesis is the two-term DNF $T_1 \vee T_2$. If at most one of the $\mathrm{MQ}(T_1)$ and $\mathrm{MQ}(T_2)$ is 1, then the situation is somewhat more complicated. We now describe each case in more detail.

Case 1:

$\mathrm{MQ}(T_1 \cap T_2) = 1$. Our initial hypothesis is the term $T_1 \cap T_2$, and REFINEDOWN may delete variables from it, but if the target has two terms REFINEDOWN will eventually return a positive counterexample x to a perhaps revised one-term hypothesis T, and $\mathrm{MQ}(T \cap x) = 0$. In this case, x must cover a target term distinct from the target term covered by T.

i: Suppose $\mathrm{MQ}(x \cap T_1) = 1$ and $\mathrm{MQ}(x \cap T_2) = 0$. Then $T_1 \cap x$ covers a different target term than that covered by T. We try revising T_1 to something in T and T_2 to something in x, and if that fails, vice versa.

ii: Suppose $\mathrm{MQ}(x \cap T_1) = \mathrm{MQ}(x \cap T_2) = 0$. Same two options considered as in 1.i, though the details of the code are different.

iii: Suppose $\mathrm{MQ}(x \cap T_1) = \mathrm{MQ}(x \cap T_2) = 1$. We now show that this case cannot arise. Recall that $\mathrm{MQ}(T \cap x) = 0$. Thus x covers two distinct terms which are also distinct from the term covered by $T_1 \cap T_2$. Contradiction.

Case 2:

$\mathrm{MQ}(T_1 \cap T_2) = 0$.

i: Consider the case where $\mathrm{MQ}(T_1) = 1 = \mathrm{MQ}(T_2)$. We call REFINEDOWN$(T_1 \vee T_2)$; straightforward arguments show that editing each T_i to the target term contained in T_i must use fewer queries than editing T_i to the target term contained in $T_{\bar{i}}$. (Here, and in the code, $T_{\bar{i}}$ refers to T_2 if $i = 1$, and to T_1 if $i = 2$.)

ii: Next, consider the case where $\mathrm{MQ}(T_1) = 1$ and $\mathrm{MQ}(T_2) = 0$. (The case where $\mathrm{MQ}(T_1) = 0$ and $\mathrm{MQ}(T_1) = 1$ is symmetric.)
We first try refining the term contained in T_1, assuming that that target term should be derived from T_1. If we get a positive counterexample, x, to this one-term hypothesis, we binary search to revise T_2 to a term contained in x.
If all of this fails, then we try instead editing T_2 to the target term contained in T_1. We then refine that term until we reach the target or get a positive counterexample, x, to this one-term hypothesis. In that case, we binary search from T_1 to the positive counterexample to get a second hypothesis term, and finally refine down the two hypothesis terms.

iii: Finally, consider the case where $\mathrm{MQ}(T_1) = \mathrm{MQ}(T_2) = 0$. We begin by getting a positive counterexample x to EQ(FALSE). In this case, x must contain at least one term of the target DNF, and *may contain both terms* of the target DNF.
In lines 29–36, we use the "try both i" construct to try revising each T_i to x.

iii.a: If x contains only one target term, and the target DNF has two terms, this means that we try revising both T_i to that one term in x (and revising $T_{\bar{i}}$ to a second term, assuming the target has two terms). At least one of the T_i will lead to a successful revision.

iii.b: If x contains both target terms, then in one or both of the two branches of the "try both i," BINARYSEARCH may find a pivot variable (one unique to one of the target terms). If so, then on that branch, BINARYSEARCH will then return *PivotFlag* and an alternative x that contains only one term, and we back up to the "try both i at line 29 and try both i again using this x.
Notice that the call to BINARYSEARCH to find the pivot uses at most $O(e \log n)$ queries, and happens only once, so the asymptotic query complexity is not increased by finding the pivot.

iii.c: The last possibility is that x contains both target terms, but that on both branches of "try both i," the binary search from $x \cap T_i$ to x finds some revised term without ever adding a pivot variable. If both branches find the same term, then at least one of the branches has edited the syntactically closer T_i to this term, and this branch of the computation will succeed.
Suppose instead that each branch found a different term; say $T_1 \cap x$ was edited to find hypothesis term S_1 and $T_2 \cap x$ was edited to find hypothesis term S_2, and $S_1 \neq S_2$. Each S_i must contain a different target term, which we will refer to as G_i.
Is it possible that the most efficient revision of $T_1 \vee T_2$ requires that T_1 is revised to G_2 and vice-versa, given this situation? In order to show that this does not happen, we describe the edits needed for each revision.
Note that any variable in $S_i \setminus T_i$ is not a pivot, but does appear in some goal term. In other words, for each i, $S_i \setminus T_i$ is contained in $G_1 \cap G_2$. In either revision (the current one or the perverse one), those variables would have had to be added to T_i. Let us consider, for each revision what other edits would be needed.
We will ignore the variables in $T_i \setminus (G_1 \cup G_2)$, since those need to be deleted in either case. Let $G'_i = G_i \setminus G_{\bar{i}}$.
The current revision requires that, for each i, we

- add $G_i \setminus T_i$ to T_i
- delete $G'_{\bar{i}} \cap T_i$ from T_i.

The perverse revision requires that, for each i, we

- add $G_{\bar{i}} \setminus T_i$ to T_i
- delete $G'_i \cap T_i$ from T_i.

Notice that $G'_i \cap T_i = G'_i$, since the only variables added to T_i to form S_i were from $G_1 \cap G_2$. Thus there are at least as many deletions in the perverse revision as there are additions in the current revision.
Next observe that $G_i \setminus T_i$ is contained in $(G_i \cap G_{\bar{i}}) \setminus T_i$ by the argument about pivots, and this is contained in $G_{\bar{i}} \setminus T_i$. Thus, the number of perverse additions is at least as great as the number of additions in the current case.
Thus, the perverse revision is perhaps worse and certainly no better than deleting from S_1 and S_2. This shows that the length of the sequence of revisions that our algorithm is making is less than or equal to the minimum revision distance from the initial theory to the target theory (i.e., from $t_1 \vee t_2$ to $G_1 \vee G_2$).

Based on the proceeding discussion of REVISEUPTOE we have the following theorem.

THEOREM 6. *Algorithm* REVISEUPTOE *will revise a 2-term monotone DNF formula needing e revisions using at most* $O(e^2 \log n)$ *queries, where n is the number of variables in the given universe.*

COROLLARY 7. *Every monotone 2-term DNF* φ *has a revision algorithm that uses* $O\left(e^3 \log n\right)$ *queries, where e is the revision distance between* φ *and the target concept in the general model of revisions.*

PROOF. The algorithm proceeds in stages. During stage s, we run REVISEUPTOE(φ, s). This must succeed in finding the target by the time $s = e$. □

5. CONCLUSIONS, FUTURE WORK

Angluin et al. [2] give a query complexity $O(m^2 n)$ algorithm for learning Horn sentences with m clauses on n variables. This work shows that, when the number of clauses/terms in a formula is sufficiently small, it is *much* easier to revise than to learn from scratch. It is possible that our Horn clause revision algorithm can be fine-tuned to shave the exponent on m, the number of clauses. However, the lower bound of Sloan and Turán discussed in Section 1.2 shows that even for a single revision, we will always need $\Omega(m)$ queries, so this result cannot be extended from *polylog* clauses to $O(n)$ clauses.

The next obvious piece of work is to extend the DNF algorithm from 2-term to $\log n$-term DNF formulas, and to Horn formula revisions that allow additions. In addition, revision algorithms for DFAs would be of great use to all of us who teach and grade automata theory.

6. REFERENCES

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

[2] D. Angluin, M. Frazier, and L. Pitt. Learning conjunctions of Horn clauses. *Machine Learning*, 9:147–164, 1992.

[3] S. Argamon-Engelson and M. Koppel. Tractability of theory patching. *Journal of Artificial Intelligence Research*, 8:39–65, 1998.

[4] A. Blum, L. Hellerstein, and N. Littlestone. Learning in the presence of finitely or infinitely many irrelevant attributes. *J. of Comput. Syst. Sci.*, 50(1):32–40, 1995. Earlier version in 4th COLT, 1991.

[5] N. Bshouty and L. Hellerstein. Attribute-efficient learning in query and mistake-bound models. *J. Comput. Syst. Sci.*, 56:310–319, 1998.

[6] R. Davis and W. Hamscher. Model-based reasoning: Troubleshooting. In H. E. Shrobe and the American Association for Artificial Intelligence, editors, *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence*, chapter 8, pages 297–346. Morgan Kaufmann, San Mateo, CA, 1988.

[7] R. Greiner. The complexity of theory revision. *Artificial Intelligence*, 107:175–217, 1999.

[8] M. Kearns, M. Li, L. Pitt, and L. Valiant. On the learnability of Boolean formulae. In *Proc. 19th Annu. ACM Sympos. Theory Comput.*, pages 285–294. ACM Press, New York, NY, 1987.

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

[10] M. Koppel, R. Feldman, and A. M. Segre. Bias-driven revision of logical domain theories. *Journal of Artificial Intelligence Research*, 1:159–208, 1994.

[11] R. A. Marcotte, M. J. Neiberg, R. L. Piazza, and L. J. Holtzblatt. Model-based diagnostic reasoning using VHDL. In J. M. Schoen, editor, *Performance and Fault Modeling with VHDL*, chapter 6, pages 304–399. Prentice Hall, Englewood Cliffs, NJ, 1992.

[12] W. Marek and M. Truszczyński. Revision programming. *Theoretical Computer Science*, 190(2):241–277, 1998.

[13] J. de Kleer, A. K. Mackworth, and R. Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56:197–222, 1992.

[14] R. J. Mooney. A preliminary PAC analysis of theory revision. In *Computational Learning Theory and Natural Learning Systems, Volume III: Selecting Good Models*, chapter 3, pages 43–53. MIT Press, 1995.

[15] D. Ourston and R. J. Mooney. Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, 66:273–309, 1994.

[16] B. L. Richards and R. J. Mooney. Automated refinement of first-order Horn-clause domain theories. *Machine Learning*, 19:95–131, 1995.

[17] E. Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, Cambridge, MA, 1983.

[18] R. H. Sloan and G. Turán. On theory revision with queries. In *Proc. 12th Annu. Conf. on Comput. Learning Theory*, pages 41–52, 1999.

[19] G. G. Towell and J. W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13:71–101, 1993.

[20] R. Uehara, K. Tsuchida, and I. Wegener. Optimal attribute-efficient learning of disjunction, parity, and threshold functions. In *Computational Learning Theory: Eurocolt '97*, pages 171–184. Springer-Verlag, 1997.

[21] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *J. ACM*, 21:168–173, 1974.

[22] S. Wrobel. Concept formation during interactive theory revision. *Machine Learning*, 14(2):169–191, 1994.

[23] S. Wrobel. First order theory refinement. In L. De Raedt, editor, *Advances in ILP*, pages 14–33. IOS Press, Amsterdam, 1995.