Journal of Machine Learning Research 6 (2005) 1919–1938 Submitted 10/04; Revised 9/05; Published 12/05

New Horn Revision Algorithms

Judy Goldsmith GOLDSMIT@CS.UKY.EDU
Department of Computer Science
University of Kentucky
773 Anderson Tower
Lexington, KY 40506-0046

Robert H. Sloan SLOAN@UIC.EDU
University of Illinois at Chicago
Department of Computer Science
851 South Morgan Street, Room 1120
Chicago, IL 60607-7053

Editor: Stefan Wrobel

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 measured resource) 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 revision operators are allowed. In this model one of the main open problems is the efficient revision of Horn formulas. Two revision algorithms are presented for special cases of this problem: for depth-1 acyclic Horn formulas, and for definite Horn formulas with unique heads.

Keywords: theory revision, Horn formulas, query learning, exact learning, computational learning theory

1. Introduction

Computationally efficient learnability has been studied in the past two decades from many angles. For example, both the PAC and query learning models have been studied, and complexity has been variously measured in terms of sample size, the number of queries, and 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 (Blum et al., 1995; Bshouty and Hellerstein, 1998).

A related notion, *efficient revision algorithms*, has been studied in machine learning, where various approaches to building systems have been considered (see, e.g., Koppel et al., 1994; Lamma et al., 2003; Ourston and Mooney, 1994; Richards and Mooney, 1995; Towell and Shavlik, 1993). Efficient revision algorithms have received some attention in learning theory as well. 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

©2005 Judy Goldsmith and Robert H. Sloan.

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, since, for many complex concepts, the only hope of learning those concepts is if a reasonably good initial approximation is available.

The notion of distance usually considered for efficient revision 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; more detailed references to the literature are given in Wrobel's overviews of theory revision (Wrobel, 1994, 1995) and also in our recent papers (Goldsmith et al., 2002, 2004b).

The theoretical study of revision algorithms was initiated by Mooney (1995) in the PAC framework, and additional theoretical work was done by Greiner (1999a,b). We have studied revision algorithms in the model of learning with equivalence and membership queries (Goldsmith et al., 2002, 2004b) and in the mistake-bound model (Sloan et al., 2003).

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[1] Disjunctive Normal Form (DNF) with a bounded number of terms when both deletion and addition type revisions are allowed, but for the practically important case of Horn formulas 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 formulas in the general revision model (deletions and additions) emerged as perhaps the main open problem posed by our previous work on revision algorithms. One of the two results presented here extends that of Doshi (2003), who gave a revision algorithm for a special case of Horn sentences he called "unique explanations," which in the terminology presented below would be the special case of depth-1 acyclic Horn sentences where the heads must all be distinct, every clause must have a head, and the heads cannot be revised. The result we give in Section 3 removes all of his restrictions concerning the heads.

1.1 Revision with Queries

In this paper, we 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 (Angluin, 1988). This is a very well-studied model (e.g., Angluin, 1987b, 1988; Angluin et al., 1992; Auer and Long, 1999; Bshouty and Hellerstein, 1998; Blum et al., 2004; Bshouty, 1995), 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 $h = c$, 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 $c(x)$.

1. A propositional logic formula is *monotone* if it contains no negations.

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.

One scenario for practical applications is 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 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.

Perhaps the most common case for practical applications 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. Consider the following case: Expert oncologist Dr. Jones is cooperating with the local computer scientists to build a model of foobaric cancer. She gives long answers to the knowledge engineers' initial open-ended questions, and countless shorter answers as they build and refine their model. These shorter questions are membership questions: "If the patient has this complex of symptoms, do you diagnose foobaric cancer?"

Finally, in the model validation phase of the work, the knowledge engineers and computer scientists proudly present scenarios and diagnoses. And Dr. Jones shakes her head and says, "No, that's not right at all. Your system will give the wrong diagnosis in these settings; reliance on this symptom is a red herring."

These latter responses are equivalence queries, complete with counterexamples.

As an aside, even theory revision via queries for formal languages may have some application. Consider Professor Doe, who is teaching, say, Automata and Formal Languages. Her difficult student presents her with an incorrect finite automaton, and demands proof that it is incorrect. She provides a counterexample, some string that the presented automaton misclassifies. It becomes clear that the student has misunderstood the problem. String by string, he queries her about membership in the desired regular language, offering periodic updates to his automaton until either it is correct, or Professor Doe discovers a prior appointment.

Note that an efficient revision algorithm is clearly in the student's best interest in this case.

1.2 Classes of Horn Formulas Considered

Horn revision is 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 by Goldsmith et al. (2004b); the kind of learnability result that we wish to extend to revision in this paper is that of Angluin et al. (1992) for propositional Horn formulas.

In this paper we present results for the revision problem outlined above: the revision of Horn formulas in the general revision model allowing both deletions and additions (more precise defi-

nitions are given in Section 2). We use the model of learning with membership and equivalence queries.

We show that one can revise two subclasses of Horn formulas with respect to both additions and deletions of variables. The new algorithms make use of our previous, deletions-only revision algorithm for Horn formulas (Goldsmith et al., 2004b) and new techniques which could be useful for the general question as well.

1.2.1 DEPTH-1 ACYCLIC HORN

Logic programming theories are often presented as Horn theories. Each clause with a head, or nonnegated variable, is interpreted as a potential justification for making the head variable true in some model of the program. These clauses are also called "definite".

In computing stable models of logic programs, it is simplest if the logic programs are stratified (Apt et al., 1988; Chandra and Harel, 1985; Van Gelder, 1988), or acyclic (Angluin, 1987a). One begins by setting all "facts," or heads without bodies, to true.[2] Then iteratively, one sets all consequences of the current true variables to true.

At each iteration, one considers only definite clauses, and only those clauses whose heads do not appear in the currently-true variables and all of whose variables are already true. These collections of clauses, or strata of a program, are themselves depth-1 Horn theories. We begin by focusing on theory revision for these simple theories.

One of our main results, Theorem 5, 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.

1.2.2 DISTINCT HEADS/UNIQUE EXPLANATIONS

In life, and in many Horn theories, there may be multiple explanations of something, or Horn clauses with the same head. Another simplification to Horn theories, other than considering individual strata, is to consider theories that provide unique explanations for each variable; that is, theories where clauses each have a distinct head. As in the stratified theories, this allows model-building to be accomplished in one pass through the theory. [Note that this definition of "unique explanation" is simpler than that of Doshi (2003). We also refer to such theories as having "distinct heads."]

But even such simple theories are subject to revision. The expert who provides a theory may fudge on explanations, including unnecessary preconditions or omitting necessary ones. Thus, our second topic in this paper is revision with queries for theories consisting of unique explanations.

We also give a revision algorithm for definite Horn formulas with distinct 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 8).

1.3 Overview of the Rest of the Paper

Preliminaries are given in Section 2, Horn formula revisions in Sections 3 and 4, and open questions in Section 5.

2. Acyclic Horn formulas have also been studied from various other points of view, including learning (Angluin, 1987a; Arimura, 1997) and computational aspects (Hammer and Kogan, 1995).

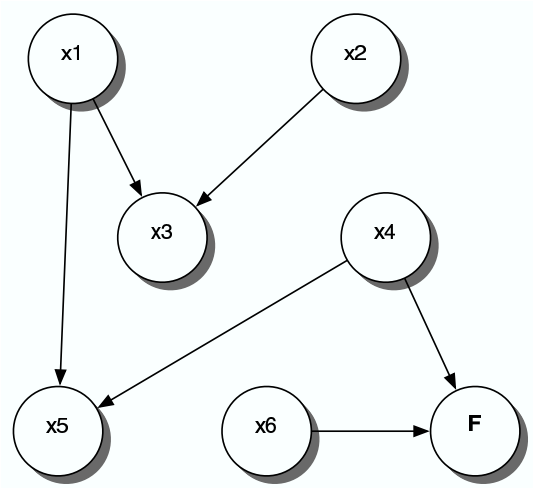

Figure 1: Graph of the Horn formula φ given by (1).

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 $\mathbf{x} \in \{0,1\}^n$. In the vocabulary of propositional logic, an instance (or vector) is a model for the target theory. 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 $\chi_Y = (\alpha_1, \ldots, \alpha_n) \in \{0,1\}^n$, where $\alpha_i = 1$ if $i \in y$ and $\alpha_i = 0$ otherwise, 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. When convenient, we treat body(c) as the vector with 1's in the positions where body(c) has variables. A Horn clause with an unnegated variable is called *definite* (or positive). If a definite clause contains only one variable, then that clause is called a *fact*. We will consider clauses with no unnegated variables to have head $\mathbf{F}$, and will sometimes write them as $(body \rightarrow \mathbf{F})$.

A *Horn formula* is a conjunction of Horn clauses. A Horn formula is definite if all its clauses are definite. A Horn formula has *unique heads* if no two clauses have the same head.

We define the *graph* of a Horn formula to be a directed graph on the variables together with $\mathbf{F}$, 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. A Horn formula is *acyclic* if its graph is acyclic; the *depth* of an acyclic Horn formula is the maximum path length in its graph (Angluin, 1987a).

For example, the Horn formula

$$\varphi = (x_1 \wedge x_2 \rightarrow x_3) \wedge x_2 \wedge (x_1 \wedge x_4 \rightarrow x_5) \wedge (x_4 \wedge x_6 \rightarrow \mathbf{F}) \tag{1}$$

is depth-1 acyclic. Its graph, shown in Figure 1, 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 $\text{head}(c) \notin \mathbf{x}$. (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, $x_1x_4 \cap 1100 = x_1$.

We use the standard model of membership and equivalence queries (with counterexamples), denoted by MQ and EQ (Angluin, 1988). In an equivalence query, the learning algorithm proposes a *hypothesis*, a formula h, and the answer depends on whether $h \equiv c$, where c is the target formula. 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 $c(\mathbf{x}) \neq h(\mathbf{x})$. If $\mathbf{x}$ is a counterexample and $c(\mathbf{x}) = 1$ and $h(\mathbf{x}) = 0$, then we refer to $\mathbf{x}$ as a positive counterexample, and otherwise a negative counterexample.

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 (including Horn sentences), 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. Note that in Algorithm 2, while we "add clauses" to a hypothesis, these are always clauses that are in the given formula but not yet in the hypothesis.) In the algorithms given in this paper, the new literals must be added to the body of a clause.

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. For DNF (resp. CNF) formulas, we 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 (Goldsmith et al., 2002).

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

Our revision algorithms use a kind of binary search, of a general kind often used in learning algorithms involving membership queries, presented as Algorithm 1. The starting points of our binary search are two instances, a negative instance **neg** and a positive instance **pos** such that **pos** $<$ **neg**. The algorithm returns a variable v that is critical in the sense that there is a (possibly empty) set S of variables from **neg** $\setminus$ **pos** such that **neg** modified by setting the variables in S to 0 is still a negative instance, but additionally setting v to 0 creates a positive instance.

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

Require: MQ(**neg**) == 0 and MQ(**pos**) == 1 and **pos** < **neg**

```
neg_0 := neg
while neg and pos differ in more than 1 position do
    Partition neg \ pos into approximately equal-size sets d_1 and d_2.
    Put mid := neg with positions in d_1 switched to 1
    if MQ(mid) == 0 then
        neg := mid
    else
        pos := mid
    end if
end while
v := the one variable on which pos and neg differ
return v
```

3. Depth-1 Acyclic Horn Formulas

We show here how to revise depth-1 acyclic Horn formulas. Depth-1 acyclic Horn formulas are precisely those where each variable that occurs as a head either occurs as a fact (the head of an empty-bodied clause) or never occurs in the body of any clause. Notice that such formulas are a class of unate CNF: variables that occur as facts are the only variables that can appear both negated and unnegated, and we can always rewrite any Horn formula with facts to a logically equivalent Horn formula where those fact variables do not appear in any clause body by using resolution. For example, φ in Equation 1 is equivalent to

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

Previously we gave a revision algorithm for unate DNF (which would dualize to unate CNF) that was presented as being able to revise specifically two clauses (Goldsmith et al., 2002). It would generalize to an algorithm whose query complexity is exponential in the number of clauses. Here we give an algorithm for an important subclass of unate CNF that is polynomial in the number of clauses.

In the following subsection we give the algorithm and its analysis; then in Section 3.2 we give an example run of the algorithm. The reader may find it helpful to switch back and forth between the two subsections.

3.1 Algorithm and Analysis

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.

Let us call those variables that occur as the head of a clause of the initial formula *head variables*. 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.

We will implicitly use the following fact often in our analysis of our algorithm.

Proposition 1 *Let h be either a variable or $\boldsymbol{F}$. If $\mathbf{x}^h$ falsifies a clause of the target depth-1 acyclic Horn formula with head h, then $\mathbf{x}$ also falsifies that clause.*

Proof Consider target clause $b \to h$, where b is nonempty and $b \to h$ is falsified by $\mathbf{x}^h$. It must be that $\mathbf{x}^h$ covers b. If $\mathbf{x}^h$ covers b, then $\mathbf{x}$ covers b, since no head variables may occur in b, and $\mathbf{x}^h \setminus \mathbf{x}$ consists only of those head variables besides h. Thus, changing those variables from 1 in $\mathbf{x}^h$ to 0 in $\mathbf{x}$ can only falsify *more* clauses. ■

The algorithm begins with an assumption that the revision distance from the initial theory to the target theory is d. If the revision fails, then d is doubled and the algorithm is repeated. Since the algorithm is later shown to be linear in d, this series of attempts does not affect the asymptotic complexity. We give a brief overview of the algorithm, followed by somewhat more detail. The pseudocode is given 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 variables. If that returns 0, then the modified $\mathbf{x}$ must falsify a headless target clause. Otherwise, for each head variable 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 $\mathbf{x}$ is associated with a head, we first try to use $\mathbf{x}$ to make deletions from each existing hypothesis clause with the same head. If no such deletions are possible, then we use $\mathbf{x}$ to add a new clause to the hypothesis. We find any necessary additions when we add a new clause.

If $(\text{body}(c) \cap \mathbf{x})^h$ (or, equivalently, $\text{body}(c)^h \cap \mathbf{x}^h$) is a negative instance, which we can determine by a membership query, then we can create a new smaller hypothesis clause whose body is $\text{body}(c) \cap \mathbf{x}$. (Notice that $\text{body}(c) \cap \mathbf{x} \subset \text{body}(c)$ because as a negative *counterexample*, $\mathbf{x}$ must satisfy c. Furthermore, since $\text{MQ}(\mathbf{x}^F) = 1$ and $\text{MQ}(\mathbf{x}^h) = 0$, we know that h is not in $\mathbf{x}$.)

To use $\mathbf{x}$ to add a new clause, we then use an idea from the revision algorithm for monotone DNF (Goldsmith et al., 2002). 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 $\mathbf{x}$ intersect $\{$the initial clause with the other heads set to 1$\}$ up to $\mathbf{x}$. If we get to something negative with fewer than d additions, we update $\mathbf{x}$ to this negative example.

Whether or not $\mathbf{x}$ 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 $\mathbf{x}$ falsifies. All necessary additions to this clause are found by the calls to BINARYSEARCH; later only deletions will be needed.

We now give a series of lemmas that will together prove the correctness and query complexity of HORNREVISEUPTOD(φ, d). 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.

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

```
1: Rewrite φ to remove any facts from other clauses' bodies
2: H := everywhere-true empty conjunction
3: while (x := EQ(H)) ≠ "Correct!" and d > 0 do
4:    h :=ASSOCIATE(x, φ)
5:    if H has at least one clause then
6:       for all clauses c ∈ H with head h do
7:          if MQ((body(c) ∩ x)^h) == 0 then {delete vars from c}
8:             body(c) = body(c) ∩ x
9:             d := d−number of variables removed
10:         end if
11:      end for
12:   end if
13:   if no vars. were deleted from any clause then {find new clause to add}
14:      FoundAClause:= false; min := d
15:      for all c ∈ φ with head h (or all c ∈ φ if h == F) do
16:         new = body(c)^h ∩ x^h
17:         numAddedVars = 0 {# additions to body for this c}
18:         while MQ(new) == 1 and numAddedVars < d do
19:            l := BINARYSEARCH(x^h, new)
20:            new := new ∪ {l}
21:            numAddedVars := numAddedVars + 1
22:            if MQ(x − {l}) == 0 then {(x − {l}) is a "pivot"}
23:               x := x − {l}
24:               restart the for all c loop with this x—go to Line 14 to reset other parameters
25:            end if
26:         end while
27:         if MQ(new) == 0 then
28:            x := new
29:            FoundAClause := true
30:            min := min(numAddedVars, min)
31:         end if
32:      end for
33:      if not FoundAClause then
34:         return"Failure"
35:      else
36:         Set all head variables of x to 0
37:         H := H ∧ (x → h) {treating x as monotone disjunction}
38:         d := d − min
39:      end if
40:   end if
41: end while
42: return H is last EQ returned "Correct!", otherwise return "Failure"
```

Algorithm 3 ASSOCIATE$(\mathbf{x}, \varphi)$

1: **if** MQ$(\mathbf{x}^{\mathbf{F}}) == 0$ **then**
2: **return F**
3: **end if**
4: **for** each head variable h that is 0 in $\mathbf{x}$ **do**
5: **if** MQ$(\mathbf{x}^h) == 0$ **then**
6: **return** h
7: **end if**
8: **end for**

Lemma 1 *Algorithm* HORNREVISEUPTOD *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 after it is initialized. 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 (at Line 7 for updates of existing hypothesis clauses and at Line 27 for adding new clauses) that 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 head variables 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 *Let negative counterexample* $\mathbf{x}$ *be associated with head* h*. If* $\mathbf{x}$ *is not used to make deletions, then* $\mathbf{x}^h$ *falsifies any target clauses with head* h *whose body is covered by* $\mathbf{x}$*. Further, if* $\mathbf{x}$ *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 If $\mathbf{x}$ falsified the same target clause as an existing hypothesis clause body with head h, then the membership query at Line 7 would return 0, and $\mathbf{x}$ would be used to delete variables from that hypothesis clause body.

Now $\mathbf{x}$ may be changed from the value it had at Line 7 before it is used to actually add a new clause. However, those changes (made when $\mathbf{x}$ is updated at Line 28) in fact change certain non-head variables of $\mathbf{x}$ from 1 to 0, so the updated $\mathbf{x}$ can falsify only fewer clauses than the original $\mathbf{x}$. Thus if and when $\mathbf{x}$ is used to add a new clause, $\mathbf{x}$ cannot falsify the same target clause as any existing hypothesis clause with the same head. The newly added hypothesis clause's body is a subset of $\mathbf{x}$, so that clause body does not cover any other hypothesis clause body with head h. ■

The next lemma is the heart of the analysis of HORNREVISEUPTOD.

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

Proof Let the initial formula be $\varphi = \bigwedge_{i=1}^{m} c_i^0$ and the target formula be $\psi = \bigwedge_{i=1}^{m'} c_i^*$. We will assume throughout the analysis in this proof that the terms of the initial and target formulas are numbered so that they "line up" for calculating the revision distance from φ to ψ. That is, the revision distance is

$$(m - m') + \sum_{i=1}^{m'} dist(c_i^0, c_i^*),$$

where $dist(c_i^0, c_i^*)$ is the revision distance from clause c_i^0 to c_i^*, and is equal to a "body distance" that is the symmetric difference between the bodies of the two clauses, plus a "head distance" that is 1 if $\text{head}(c_i^*) = \mathbf{F}$ and $\text{head}(c_i^0) \neq \mathbf{F}$, and 0 if $\text{head}(c_i^*) = \text{head}(c_i^0)$ (and is infinite in any other case). Note that $m - m'$ accounts for the clauses deleted and that $m \geq m'$ because we cannot add entirely new clauses.

Let d_r be the value of the variable d at the start of the rth iteration of the outer **while** loop. We argue by induction on r both that d_r is an upper bound on the number of revisions required to get from a formula made of those terms in the current hypothesis and the remaining terms in the initial formula to the target, and that the rth iteration does not fail.

More precisely, assume that at the start of round r, the hypothesis H_r is $c_1 \wedge c_2 \wedge \cdots \wedge c_{\ell_r}$. Part of our inductive claim is that there is a map $a(i)$ (technically a relation) of hypothesis clauses to target clauses such that

$$(\text{body}(c_i))^{\text{head}(c_i)} \text{ falsifies target clause } c^*_{a(i)}. \tag{2}$$

Formally a is a relation because some hypothesis clauses may be mapped to more than one target clause; that will occur precisely when (2) holds for more than one target clause. The relation a maps every index i of a hypothesis clause to at least one target clause index, and is one-to-one in the sense that no two target clauses ever have the same hypothesis clause mapped to both of them. The relation a evolves in only two ways: (1) when $a(i)$ is more than one index, sometimes one of those indices gets dropped, and (2) a new i gets added to the domain of a each time a clause is added to the hypothesis. For convenience of notation, we will somewhat sloppily refer to $c^*_{a(i)}$ as if it were one clause, when we mean that such statements hold for each of the associated target clauses.

The rest of the inductive claim is that: (i) the rth iteration of HORNREVISEUPTOD does not fail, and (ii) at the start of iteration r of HORNREVISEUPTOD,

$$d_r \geq \sum_{c_i \in H} \left| \text{body}(c_i) \setminus \text{body}(c^*_{a(i)}) \right| + \sum_{c_j \notin H} dist(c_j^0, c_j^*). \tag{3}$$

For the base case, $d_1 = d$, hypothesis H_1 has no terms, and Equation (3) is satisfied, since the right hand side is the revision distance from φ to ψ less $(m - m')$.

To complete the base case, we must argue that the first iteration does not fail. We start with a counterexample $\mathbf{x}$ that is associated with head h. We need to show that a new clause is found using $\mathbf{x}$ by at least one iteration of the **for all** c loop starting at Line 15.Let c_i^* be a target clause with head h or $\mathbf{F}$ that $\mathbf{x}$ falsifies. At some point c_i will be used as the clause in the **for all** $c \in \varphi$ loop at Line 15. As long $\mathbf{x}^h$ falsifies *only* target clause c_i^*, then after at most d calls to BINARYSEARCH, all necessary additions to c_i will have been found and a clause will be added, completing the base case. (Even if $\mathbf{x}^h$ falsifies multiple target clauses, this still might happen.)

However, if $\mathbf{x}^h$ falsifies more than one target clause, then we may find a variable that appears to be a necessary addition but is really a necessary addition to a different clause. Fortunately, this requires only one query to verify (see Line 22 of the algorithm). When such a variable (a "pivot")

is found, we set that variable of $\mathbf{x}$ to off so that the new value of $\mathbf{x}^h$ falsifies fewer clauses. Thus, this can occur at most $m-1$ times before $\mathbf{x}$ falsifies exactly one clause, and no more pivots may be found. Once that happens, we must find a clause.

For the inductive step, there are two cases.

Case 1: $\mathbf{x}^h$ is not used to delete any variables from any target clause. The argument that this iteration does not fail is the same as the corresponding argument for the base case.

As in the base case, there may be some number of times that a pivot is found and set to 0 in $\mathbf{x}$. Now consider the value of $\mathbf{x}$ after any pivots have been found, and after the last time $\mathbf{x}$ is updated at Line 27. By Lemma 2, $\mathbf{x}$ does not cover any target clauses covered by clause bodies in the hypothesis, so it must cover one or more new target clauses. Let c_j^* be one of those target clauses. The "body" revision distance $dist(c_j^0, c_j^*)$ is equal to the number of "necessary additions", $\left|\text{body}(c_j^*) \setminus \text{body}(c_j^0)\right|$, plus the number of "necessary deletions", $\left|\text{body}(c_j^0) \setminus \text{body}(c_j^*)\right|$. In the iteration of the **for all** c loop at Line 15 with c set to c_j^0, all the necessary additions had to be found, and the value of *numAddedVars* for that iteration would have been the number of the necessary additions, so at most that number is subtracted from d_r. Also, $\mathbf{x}$ after that intersection contained at most the variables in the body of c_j^* before the necessary deletions are made. In later revisions, all that can happen is some of those necessary deletions might happen to be made. Thus Equation (3) holds at the end of the rth iteration of the outer **while** loop. To complete the inductive step for this case, note that the relation a can indeed be extended by relating the index of the new hypothesis clause to the one or more target clauses whose body its body covers, so Equation (2) holds.

Case 2: $\mathbf{x}^h$ is used to delete variables from at least one hypothesis clause. Say deletions are made to hypothesis clause c_i. Now $(\text{body}(c_i) \cap \mathbf{x})^h$ can falsify only the same or fewer clauses than $\text{body}(c_i)$ falsifies. By the inductive hypothesis (specifically Equation (2) coupled with Proposition 1), $\text{body}(c_i)$ falsifies target clause(s) $c_{a(i)}^*$. Thus the updated hypothesis clause $c_i := (\text{body}(c_i) \cap \mathbf{x})$ must falsify some or all of the clause(s) $c_{a(i)}^*$, and the relation a is either unchanged, or altered by decreasing the range of $a(i)$. Equation (3) still holds because we decrease d_r by the number of deletions we make, and we also decrease $|\text{body}(c_i) \setminus \text{body}(c_a^*(i))|$ by the number of deletions we make.

Clause $c_{a(i)}^*$ could be derived from c_i by deletion edits; that is, $\text{body}(c_i)$ falsifies $c_{a(i)}^*$. By Proposition 1, since $\text{MQ}((\text{body}(c_i) \cap \mathbf{x})^h) = 0$, it must be that $\text{body}(c_i) \cap \mathbf{x}$ falsifies a target clause with head h. Further, using the numbering of the target clauses that makes that target clause correspond to c_i at the start of the round, the number of variables removed from $\text{body}(c)$ is subtracted from the parameter d, and Equation (3) still holds, completing the induction step.

We will find all the necessary additions to $\text{body}(c_i)$ using at most d_r calls to BINARYSEARCH (in fact, using at most $|\text{body}(c_i^*) \setminus \text{body}(c_i)|$) calls. Furthermore, the clause added will have all the variables in $\text{body}(c_i^*)$ and no variables not in $\text{body}(c_i) \cup \text{body}(c_i^*)$ (i.e., it will need at most only necessary deletion revisions), and the parameter d_r will be decreased by at most the number of added variables. ■

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

Proof If the variable d ever becomes nonpositive, then we terminate the algorithm.

ASSOCIATE makes at most m equivalence queries per negative counterexample. Next we try to use negative counterexample $\mathbf{x}$ to make deletions from an existing clause. This consumes exactly 1 equivalence query and at most m membership queries. If any deletions are made, we decrease d by at least 1.

There are at most d such counterexamples used for deletions. Each counterexample used for deletions uses $\leq m+1$ queries.

If a counterexample is not used for deletions, then we use it to add a new clause. We can have at most $m-1$ restarts (where we back up to Line 2) due to "pivots." These occur when $\mathbf{x}$ falsifies multiple clauses, and each time one is found, $\mathbf{x}$ is modified so that it falsifies fewer clauses.

There are at most m restarts, and ignoring the restarts, the main **forall** loop at Line 15 iterates over at most all m initial theory clauses. For each one iteration, the inner **while** loop iterates at most d times (once for each added literal). Each iteration of that inner **while** loop makes two direct membership queries, and one call to BINARYSEARCH, which uses at most $\log n$ queries.

Thus, each of $\leq m$ (re)starts uses at most $m \cdot d \cdot \log n$ queries, plus $2m+1$ queries to establish that the particular counterexample should be used for the addition of a clause.

Thus the algorithm HORNREVISEUPTOD(φ, d) correctly revises initial formula φ using $O(d \cdot (m+1) + m \cdot m^2 \cdot d \cdot \log n) = O(m^3 \cdot d \cdot \log n)$ queries. ■

Theorem 5 *There is a revision algorithm for depth-1 acyclic Horn formulas with query complexity $O(d \cdot m^3 \cdot \log n)$, where d is the revision distance, n is the number of variables in the universe, and m is the number of clauses in the initial formula.*

Proof Lemmas 3 and 4 together have shown the desired theorem. ■

3.2 An Example Run of HORNREVISEUPTOD

We now give an example run of HORNREVISEUPTOD. Suppose the variable set is $\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$ and the initial formula φ and the target formula ψ are given by

$$\begin{aligned} \varphi &= x_2 \wedge (x_1 \rightarrow x_3) \wedge (x_1 \wedge x_4 \rightarrow x_5) \wedge (x_4 \wedge x_6 \rightarrow \mathbf{F}) \\ \psi &= x_2 \wedge (x_1 \rightarrow \mathbf{F}) \wedge (x_4 \wedge x_6 \wedge x_7 \wedge x_8 \rightarrow x_5). \end{aligned} \tag{4}$$

The revision distance from φ to ψ is 5: 1 for the deletion of head x_3 from second clause, 1 for the deletion of the third clause, and 3 for adding the literals $\bar{x}_7$, $\bar{x}_8$, and x_5 to the fourth clause (i.e., adding x_7 and x_8 to the body of the fourth clause, and x_5 to the head of the fourth clause).

For future reference, the head variables in the initial theory are x_2, x_3 and x_5.

Assume now that Algorithm HORNREVISEUPTOD is called with inputs φ and any $d \geq 5$. It initializes its hypothesis H to the everywhere true empty conjunction. Assume EQ(H) returns $\mathbf{x} = 11101110$, a negative counterexample. Now we call ASSOCIATE$(\varphi, 11101110)$ to find a candidate head for a clause negated by 11101110. In ASSOCIATE we immediately find that MQ$(11101110^{\mathbf{F}})$ = MQ$(11101110) = 0$, so ASSOCIATE$(\varphi, 11101110)$ returns **F**. (Recall that the operation $\mathbf{x}^{\mathbf{F}}$ sets all head variables of $\mathbf{x}$ to 1.)

Hypothesis H currently has no clauses, so we will use 11101110 to add a new clause to H starting at Line 13. Because ASSOCIATE returned **F**, each of the four clauses of φ is considered. Say they're processed in the order they are written in Equation (4). Starting with x_2, we set **new** to be $\text{body}(x_2)^{\mathbf{F}} \cap 11101110^{\mathbf{F}} = 01101000 \cap 11101110 = 01101000$. That is a positive instance, so we begin making calls of BINARYSEARCH from $\mathbf{x}^{\mathbf{F}} = 11101110$ to **new**. Now BINARYSEARCH$(11101110, 01101000)$ returns the position x_1. We turn position x_1 to 1 in **new**, so now **new** is 11101000, and increment *numbAddedVars* to be 1 instead of 0. Turning position x_1 to 0 in 11101110 yields a positive instance, so we do *not* have a pivot (Lines 22–25). Now $\text{MQ}(\mathbf{new}) = 0$, so we update **x** to be 11101000, and set *FoundAClause* to true and *min* to *numbAddedVars* , which is 1.

Now we have to consider the next three clauses of φ. However, when we intersect $\mathbf{x}^{\mathbf{F}} = 11101000$ with $\text{body}(c)^{\mathbf{F}}$ at Line 15 for each of the remaining three clauses c in φ we get back **x**, so no changes are made.

Thus in Lines 36–38, we update H to be

$$H = (x_1 \rightarrow \mathbf{F}),$$

and decrement d by 1, and begin the next iteration of the outer **while** loop by making another equivalence query.

Say this time we receive the negative counterexample $\mathbf{x} = 01110111$. When we call ASSOCIATE, the instance $01110111^{\mathbf{F}} = 0111111$ is positive, so **F** is *not* returned. The only head variable in 01110111 that is 0 is x_5, and $\text{MQ}(01110111^{x_5}) = \text{MQ}(01110111) = 0$, so ASSOCIATE returns $h = x_5$. There is no clause in H with head x_5, so we do not try to use instance 01110111 to delete variables from any clause of H.

In the **for** loop starting at Line 13 we consider only clauses with head x_5; there is exactly one: $c = (x_1 \wedge x_4 \rightarrow x_5)$. We set $\mathbf{new} = \text{body}(c)^{x_5} \cap 01110111^{x_5} = 10010000^{x_5} \cap 01110111^{x_5} = 11110000 \cap 01110111 = 01110000$, which is a positive example.

Again, we make calls of BINARYSEARCH from $\mathbf{x}^h = 01110111$ to **new**. Assume that the first call returns position x_8. Then we update **new** to 01110001, and *numAddedVars* to 1. Since **new** is still a positive instance, we call BINARYSEARCH again. Say this time it returns position x_7. We update **new** to 01110011, and *numAddedVars* to 2. Instance **new** remains positive; we call BINARYSEARCH again; it returns x_6; we update **new** to 01110111 and *numAddedVars* to 3. Finally **new** is a negative instance, so we update $\mathbf{x} = \mathbf{new} = 01110111$, set *FoundAClause* to true and *min* to 3.

In Lines 36–38 we set all head variables of **x** to 0 so $\mathbf{x} = 00010111$ and add a new clause of the form $\mathbf{x} \rightarrow h$ to H; thus we update H to be

$$H = (x_1 \rightarrow \mathbf{F}) \wedge (x_4 \wedge x_6 \wedge x_7 \wedge x_8 \rightarrow x_5),$$

and decrement d by 3, so d has now been reduced by 4 altogether.

We begin our next iteration of the outer loop by receiving the counterexample $\mathbf{x} = 00000000$ in response to $\text{EQ}(H)$. When we call ASSOCIATE$(\mathbf{x})$, it determines that $00000000^{\mathbf{F}} = 01101000$ is a positive example, and so does not return **F**, and that $00000000^{x_2} = 00101000$ is a negative example, and so does return $h = x_2$. There is no clause in H with head x_2, so we do not try to use **x** to delete variables from existing clauses of H.

Instead, we again execute the **for** loop starting at Line 13. This time we consider only the one clause $c = x_2$ with head x_2 (and empty body). We set **new** $= \text{body}(x_2)^{x_2} \cap \mathbf{x}^{x_2} = 000000000^{x_2} \cap 00000000^{x_2} = 00101000$, which is a negative instance. Thus the **while** loop at Lines 18–26 is not executed at all. We skip over it and set all head variables of $\mathbf{x}$ to 0; thus $\mathbf{x} = 00000000$. We update the hypothesis to

$$H = (x_1 \rightarrow \mathbf{F}) \wedge (x_4 \wedge x_6 \wedge x_7 \wedge x_8 \rightarrow x_5) \wedge x_2.$$

The variable *numAddedVars* was 0; so *min* was 0, and d is not changed from its previous value (4 less than its initial value).

Now H is the target formula, so a final equivalence query returns "Correct!" This simple example did not by any means exercise every path through the algorithm's pseudocode, but it should give the general idea.

4. Definite Horn Formulas with Unique Heads

We give here a revision algorithm for definite Horn formulas with unique heads. A revision of a formula from class $\mathcal{C}$ must also be in class $\mathcal{C}$, so in particular, a revision of a definite Horn formula also be a definite Horn formula. Thus head variables cannot be fixed to 0. We use the algorithm for revising Horn formulas in the deletions-only model presented by Goldsmith et al. (2004b) as a subroutine. Its query complexity is $O(d \cdot m^3 + m^4)$, where d is the revision distance and m is the number of clauses in the initial formula.

For this algorithm we again first give the algorithm and its analysis, and then in Section 4.2 give an example run of (the main part of) the algorithm.

Algorithm 4 DEFINITEHORNREVISE(φ). Revises φ, a definite Horn formula with unique heads

```
1:  H := everywhere-true empty conjunction
2:  for all clauses c = (b → h) of φ do
3:     0_h := vector with 0 at h, 1's elsewhere
4:     if MQ(0_h) == 0 then
5:        x := vector with a 1 for every variable in b and every head of a clause of φ except h, and
          0's elsewhere
6:        while MQ(x) ≠ 0 do
7:           v := BINARYSEARCH(0_h, x)
8:           Add variable v to clause body b
9:           Set position v to 1 in x
10:       end while
11:       Add all heads of φ except h to b
12:       H := H ∧ (b → h)
13:    end if
14: end for
15: return DELETIONSONLYREVISE(H)
```

Our algorithm, DEFINITEHORNREVISE(φ), presented as Algorithm 4, has a first phase that both deletes any clauses that need deleting in their entirety and 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 (Goldsmith et al., 2004b) for revising Horn formulas in the deletions-only model of revision.

For each clause $c = (b \rightarrow h)$, the check in Line 4 whether the vector that is 0 at h and 1 elsewhere is a negative instance determines whether clause c should be deleted altogether.

To find all necessary additions to the body b of clause $c = (b \rightarrow h)$, we use a constructed example $\mathbf{x}_c$. We initialize $\mathbf{x}_c$ to b^h (the body variables from b, plus all head variables except h). Notice that the only way $\mathrm{MQ}(\mathbf{x}_c)$ can be 0 is if $\mathbf{x}$ covers the body of a clause but not its head. Since $\mathbf{x_c}$ includes all heads except h, it is clear *which* clause body is or is not covered by $\mathbf{x_c}$; the notion of "pivots" is not needed in this algorithm.

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, and $b \rightarrow h$ is added to the hypothesis. 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, a clause with head h and a body variable corresponding to each 1 in $\mathbf{x}_c \rightarrow h$ is added to the hypothesis.

Once the necessary additions to every clause in the initial theory are found, a Horn formula needing only deletions has been produced, and the deletions-only algorithm DELETIONSONLYREVISE from (Goldsmith et al., 2004b) is used to complete the revisions.

Notice that each $\mathbf{x}_c$ is generated, and each clause is added to the hypothesis, without any equivalence queries being asked. Thus, all additions may be made before any deletions are considered.

4.1 Analysis

The key part of the analysis of the revision complexity of this algorithm is the analysis of the initial processing of each clause. First we show that any entire clause deletions are correct, then we consider the addition of variables to an initial clause.

Lemma 6 *Algorithm* DEFINITEHORNREVISE *adds a clause that is either the initial formula clause c itself, or a revision of initial clause c made by adding variables to body(c), at Line 12 if and only if some revision of c appears in the target formula.*

Proof Let $c = (b \rightarrow h)$. Vector $\mathbf{0}_h$ is 0 at position h and 1 elsewhere. If any clause that is a revision of c appears in the target (not counting the everywhere true clause, which can be omitted from any conjunction), then $\mathbf{0}_h$ must falsify this target clause. In this case, a revision of c is added to the algorithm's hypothesis.

Conversely, if $\mathbf{0}_h$ is a positive instance, then it must be that the target contains no clause with head h, and hence, since the formulas are definite Horn formulas with unique heads, no clause that is a revision of c. In this case, the algorithm does not add any clause that is a revision of c to its hypothesis. ■

Lemma 7 *If any variable is added to the body of an initial clause c of φ in Algorithm* DEFINITEHORNREVISE(φ)*, then some clause c_* that is derived from c must be in the target formula, and every variable added to c in the loop in Lines 6–9 must be in c_*.*

Proof If variables are added to the body of clause c, then eventually a clause is added to the hypothesis, and by Lemma 6, we know that this means that a clause derived from c must be in the target formula.

Variable v is added to body(c) in the loop at Lines 6–9 only if there is a point in the computation where there are instances $\mathbf{x}$ and $\mathbf{x}'$ such $\mathbf{x}$ is a positive instance and $\mathbf{x}'$ is a negative instance, and $\mathbf{x}'$ is $\mathbf{x}$ with position v, and possibly some other positions that are not the head of any clause, changed from 0 to 1. Furthermore, $\mathbf{x}'$ with position v set to 0 is a positive instance. By the construction, both $\mathbf{x}$ and $\mathbf{x}'$ must have a 1 in the position of every head except for the head h of c, so $\mathbf{x}'$ must falsify a target clause that is a revision of c. Furthermore, since $\mathbf{x}'$ with v set to 0 is a positive instance, v must be in that target clause. ■

From those two lemmas we can prove:

Theorem 8 *There is a revision algorithm for definite Horn formulas with unique heads in the general model of revision with query complexity $O(m^5 + d \cdot m^3 + d \cdot \log n)$, where d is the revision distance from the initial formula to the target formula, m is the number of clauses in the initial formula, and n is the number of variables in the universe.*

Proof By Lemmas 6 and 7, each variable added to a clause is necessary, and any clause deleted in the **for** loop is unnecessary.

The query complexity for the necessary additions is at most $O(\log n)$ per added variable, which contributes a factor of $O(d \log n)$.

Algorithm DELETIONSONLYREVISE has complexity $(m^4 + d \cdot m^3)$ (Goldsmith et al., 2004b), where m is the number of clauses in the formula to be revised, and d is the revision distance. Now the formula to given to Algorithm DELETIONSONLYREVISE has revision distance at most $d + m(m-1)$, where the $m(m-1)$ comes from the up to $m-1$ heads added to the bodies of up to m clauses. Combining this information, we get a final query complexity of $O(m^5 + d \cdot m^3 + d \log n)$. ■

4.2 An Example Run of DEFINITEHORNREVISE

We present an example run of DEFINITEHORNREVISE. Suppose the variable set is $\{x_1, x_2, x_3, x_4, x_5, x_6\}$ and the initial formula φ and the target formula ψ are given by

$$\begin{aligned} \varphi &= x_1 \wedge (x_5 \wedge x_6 \rightarrow x_2) \wedge (x_2 \wedge x_3 \wedge x_4 \rightarrow x_5) \wedge (x_2 \rightarrow x_6) \\ \psi &= (x_3 \rightarrow x_1) \wedge (x_5 \wedge x_6 \rightarrow x_2) \wedge (x_3 \wedge x_4 \wedge x_6 \rightarrow x_5) \end{aligned} \tag{5}$$

The revision distance from φ to ψ is 4: 1 for adding x_3 to the body of the first clause, 2 for adding one variable and deleting another from the body of the third clause, and 1 for deleting the fourth clause.

We process the four clauses of φ in order:

1. *Clause x_1*: The vector $\mathbf{0}_{x_1} = 011111$ (i.e., 0 only at x_1) is a negative instance, so we retain this clause. We set $\mathbf{x}$ to 010011, which is a positive instance, so we enter the **while** loop of Algorithm DEFINITEHORNREVISE at Lines 6–10. In the first iteration BINARYSEARCH$(\mathbf{0}_{x_1}, \mathbf{x})$

returns x_3. Setting x_3 to 1 in $\mathbf{x}$ makes $\mathbf{x} = 011011$, which is a negative instance, so we are done with the **while** loop.

We insert the clause

$$(x_2 \wedge x_3 \wedge x_5 \wedge x_6 \rightarrow x_1)$$

into the hypothesis.

2. *Clause* $(x_5 \wedge x_6 \rightarrow x_2)$: Vector $\mathbf{0}_{x_2} = 101111$ is negative, so we retain this clause. We set $\mathbf{x} = 100011$, which is a negative instance, so we do not need to call BINARYSEARCH.

   We insert the clause

   $$(x_1 \wedge x_5 \wedge x_6 \rightarrow x_2)$$

   into the hypothesis.

3. *Clause* $(x_2 \wedge x_3 \wedge x_4 \rightarrow x_5)$: Vector $\mathbf{0}_{x_5} = 111101$ is negative, so we retain this clause. We set $\mathbf{x} = 111101$, which is a negative instance, so we do not need to call BINARYSEARCH.

   We insert the clause

   $$(x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_6 \rightarrow x_5)$$

   into the hypothesis.

4. *Clause* $(x_2 \rightarrow x_6)$: Vector $\mathbf{0}_{x_6} = 111110$ is positive, so we do not put this clause in the hypothesis (i.e., we delete this clause).

Our hypothesis is now

$$H = (x_2 \wedge x_3 \wedge x_5 \wedge x_6 \rightarrow x_1) \wedge (x_1 \wedge x_5 \wedge x_6 \rightarrow x_2) \wedge (x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_6 \rightarrow x_5).$$

We now have a hypothesis needing only deletion edits, and we pass this hypothesis to the deletions-only algorithm from Goldsmith et al. (2004b).

5. Conclusions and Open Questions

Horn formulas are ubiquitous in Computer Science, occurring in subfields from expert systems to databases. In all of these instances, the formulas or theories are dependent on human expertise or on potentially changing conditions. In many cases, "oracles" capable of answering equivalence or membership queries are far easier to come by than are direct sources for the correct theories.

The problem of revising Horn formulas with queries remains open, but this paper has broken the additions barrier. Open questions range from small to large improvements on the results presented here. For instance, we have presented a revision algorithm for acyclic, depth-1 Horn formulas. Can the techniques used here be extended to acyclic depth-2 or depth-k formulas? For acyclic formulas with unbounded depth? How much harder is it to revise Horn formulas with unique heads if we allow up to one occurrence of **F** as a head? Bounded or unbounded occurrences?

If we can revise acyclic depth-k Horn formulas for each k, how does the complexity depend on k? Is the problem fixed-parameter tractable? Could the complexity of revision depend on a fixed maximum number of occurrences as the head of a clause for each variable?

Acknowledgments

This work was partially supported by NSF grants CCR-0100040 and ITR-0325063 to the first author, and NSF grants CCR-0100336 and CCF-0431059 to the second author. The authors also thank Jignesh Doshi for inspiring this work, and Balázs Szörényi and György Turán for publishing their results with a preliminary version of the results given here, in a joint ALT'04 paper (Goldsmith et al., 2004a).

References

Dana Angluin. Learning propositional Horn sentences with hints. Technical Report YALEU/DCS/RR-590, Department of Computer Science, Yale University, December 1987a.

Dana Angluin. Learning regular sets from queries and counterexamples. *Inform. Comput.*, 75(2): 87–106, November 1987b.

Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

Dana Angluin, Michael Frazier, and Leonard Pitt. Learning conjunctions of Horn clauses. *Machine Learning*, 9:147–164, 1992.

Krzysztof R. Apt, Howard Blair, and Adrian Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, Los Altos, CA, 1988.

Hiroki Arimura. 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*, pages 432–445. Springer, 1997.

Peter Auer and Philip M. Long. Structural results about on-line learning models with and without queries. *Machine Learning*, 36(3):147–181, 1999.

Avrim Blum, Lisa Hellerstein, and Nick 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.

Avrim Blum, Jeffrey Jackson, Tuomas Sandholm, and Martin Zinkevich. Preference elicitation and query learning. *Journal of Machine Learning Research*, 5:649–667, 2004.

Nader Bshouty. Exact learning Boolean function via the monotone theory. *Information and Computation*, 123:146–153, 1995.

Nader Bshouty and Lisa Hellerstein. Attribute-efficient learning in query and mistake-bound models. *J. of Comput. Syst. Sci.*, 56(3):310–319, 1998.

Ashok K. Chandra and David Harel. Horn clause queries and generalizations. *Journal of Logic Programming*, 2:1–15, 1985.

Jignesh Umesh Doshi. Revising Horn formulas. Master's thesis, Dept. of Computer Science, University of Kentucky, 2003.

Judy Goldsmith, Robert H. Sloan, and György Turán. Theory revision with queries: DNF formulas. *Machine Learning*, 47(2/3):257–295, 2002.

Judy Goldsmith, Robert H. Sloan, Balázs Szörényi, and György Turán. New revision algorithms. In *Algorithmic Learning Theory, 15th International Conference, ALT 2004, Padova, Italy, October 2004, Proceedings*, volume 3244 of *Lecture Notes in Artificial Intelligence*, pages 395–409. Springer, 2004a.

Judy Goldsmith, Robert H. Sloan, Balázs Szörényi, and György Turán. Theory revision with queries: Horn, read-once, and parity formulas. *Artificial Intelligence*, 156:139–176, 2004b.

Russell Greiner. The complexity of theory revision. *Artificial Intelligence*, 107:175–217, 1999a.

Russell Greiner. The complexity of revising logic programs. *J. Logic Programming*, 40:273–298, 1999b.

Peter L. Hammer and Alexander Kogan. Quasi-acyclic propositional Horn knowledge bases: optimal compression. *IEEE Trans. Knowl. Data Eng.*, 7:751–762, 1995.

Moshe Koppel, Ronen Feldman, and Alberto Maria Segre. Bias-driven revision of logical domain theories. *Journal of Artificial Intelligence Research*, 1:159–208, 1994.

Evelina Lamma, Fabrizio Riguzzi, and Luís Moniz Pereira. Belief revision via Lamarckian evolution. *New Generation Computing*, 21:247–275, 2003.

Raymond J. Mooney. A preliminary PAC analysis of theory revision. In Thomas Petsche, editor, *Computational Learning Theory and Natural Learning Systems*, volume III: Selecting Good Models, chapter 3, pages 43–53. MIT Press, 1995.

Dirk Ourston and Raymond J. Mooney. Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, 66:273–309, 1994.

Bradley L. Richards and Raymond J. Mooney. Automated refinement of first-order Horn-clause domain theories. *Machine Learning*, 19:95–131, 1995.

Robert H. Sloan, Balázs Szörényi, and György 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, USA, August 24-27, 2003, Proceedings*, volume 2777 of *Lecture Notes in Artificial Intelligence*, pages 625–639. Springer, 2003.

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

Allen Van Gelder. Negation as failure using tight derivations for general logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 149–176. Morgan Kaufmann, Los Altos, CA, 1988.

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

Stefan Wrobel. *Concept Formation and Knowledge Revision*. Kluwer, 1994.