Pac-learning Nondeterminate Clauses

William W. Cohen
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974
wcohen@research.att.com

Abstract

Several practical inductive logic programming systems efficiently learn "determinate" clauses of constant depth. Recently it has been shown that while nonrecursive constant-depth determinate clauses are pac-learnable, most of the obvious syntactic generalizations of this language are not pac-learnable. In this paper we introduce a new restriction on logic programs called "locality", and present two formal results. First, the language of nonrecursive clauses of constant locality is pac-learnable. Second, the language of nonrecursive clauses of constant locality is strictly more expressive than the language of nonrecursive determinate clauses of constant depth. Hence, constant-locality clauses are a pac-learnable generalization of constant-depth determinate clauses.

Introduction

An active area of research is "inductive logic programming", or learning logic programs from examples. Several practical inductive logic programming systems, including GOLEM [Muggleton and Feng, 1992], FOIL [Quinlan, 1990; Quinlan, 1991], LINUS [Lavrač and Džeroski, 1992] and GRENDEL [Cohen, 1992; Cohen, 1993c] incorporate algorithms that efficiently learn programs of "determinate" clauses of constant depth. Following these experimental results, a number of formal results have also been obtained about the learnability of determinate clauses: in particular Džeroski, Muggleton and Russell [1992] showed that nonrecursive constant-depth determinate clauses are pac-learnable, and Cohen [1993b] extended this result to linear "closed" recursive constant-depth determinate clauses.

Unfortunately, most generalizations of this language are not pac-learnable. Given certain cryptographic assumptions, the languages of constant-depth determinate clauses with arbitrary recursion, nonrecursive log-depth determinate clauses, and nonrecursive constant-depth indeterminate clauses are *not* pac-learnable [Cohen, 1993a]. These formal results are disappointing, as they suggest that efficient general-purpose learning algorithms for non-determinate or arbitrary-depth clauses may be difficult to find.

In this paper, we introduce a restriction on logic programs called *locality*, and present two new formal results. First, we show that clauses of constant locality are pac-learnable. Second, we show that the language of clauses of constant locality is strictly more expressive than the language of determinate clauses of constant depth. Hence, the language of constant-locality clauses is a pac-learnable generalization of the language of constant-depth determinate clauses.

Formal Preliminaries

Pac-learnability: Basic Definitions

For readers unfamiliar with formal learning theory, we give below a brief overview of our learning model.

Let X be a set, called the *domain*, and define a *concept* C over X to be a representation of some subset of X. A *language* LANG is defined to be a set of concepts. Associated with X and LANG are two *size complexity measures*; we will use X_n (respectively LANG_n) to represent the set of all elements of X (respectively LANG) of size complexity no greater than n. An *example of* C is a pair (x, b) where $b =$"+" if $x \in C$ and $b =$"−" otherwise. If D is a probability distribution on X, then a *sample of* C *from* X *drawn according to* D is a pair of multisets S^+, S^- drawn from X according to D, with S^+ containing the positive examples of C and S^- containing the negative examples.

The model of *pac-learnability* was first introduced by Valiant [1984]. A language LANG is *pac-learnable* iff there is an algorithm $\text{PACLEARN}(S^+, S^-, \epsilon, \delta)$ and a polynomial function $m(\frac{1}{\epsilon}, \frac{1}{\delta}, n_e, n_t)$ so that for every $n_t > 0$, every $n_e > 0$, every $C \in \text{LANG}_{n_t}$, every $\epsilon : 0 < \epsilon < 1$, every $\delta : 0 < \delta < 1$, and every probability distribution function D, for any sample S^+, S^- of C from X_{n_e} drawn according to D containing at least $m(\frac{1}{\epsilon}, \frac{1}{\delta}, n_e, n_t)$ examples,

- PACLEARN outputs a hypothesis H such that $Prob(D(H - C) + D(C - H) > \epsilon) < \delta$

- PACLEARN runs in time polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, n_e, n_t, and the number of examples, and
- the hypothesis H is in LANG.

Notice that n_e represents the size of the examples, and n_t represents the size of the "target concept" C.

Logic Programs

For readers unfamiliar with logic programs, we give below a brief overview of their syntax and semantics. Our presentation is simplified as we are interested in the restricted case of non-recursive function-free single-clause Prolog programs. For this case our definitions coincide with the usual semantics of Prolog programs [Lloyd, 1987].

A *database DB* is a set of *(ground) facts*, each of which is of the form $p_i(t_{i_1}, \ldots, t_{i_{k_i}})$, where p_i is a *predicate symbol*, the t_{i_j}'s are *constant symbols* and k_i is the *arity* of the fact. We will usually write constant and predicate symbols as lower-case strings, such as "mary" or "sister".

Logic programs also include *variables*, which we will usually represent with capital letters such as X and Y. A *substitution* is a partial function mapping variables to constant symbols and variables; we will use the Greek letters θ and σ for substitutions, or write them as sets $\theta = \{X_1 = s_1, X_2 = s_2, \ldots, X_n = s_n\}$ where s_i is the constant (or variable) onto which X_i is mapped. A *literal* is written $p(X_1, \ldots, X_k)$ where p is a predicate symbol and $X_1, \ldots, X_k$ are variables. If θ and σ are substitutions and A is a literal, we will use $A\theta$ to denote the result of replacing each variable X in A with the constant symbol to which X is mapped by θ. We abbreviate $(A\theta)\sigma$ as $A\theta\sigma$.

A *(definite) clause* is written $A \leftarrow B_1, \ldots, B_l$ where A and $B_1, \ldots, B_l$ are literals. A is the *head* of of the clause, and $B_1, \ldots, B_l$ is the *body*. Finally, if the *extension of a clause* $C = (A \leftarrow B_1, \ldots, B_l)$ *with respect to a database DB*, written $ext(C, DB)$, is the set of all facts f such that either

- $f \in DB$, or
- there exists a substitution θ so that $A\theta = f$, and for every B_i in the body of the clause, $B_i\theta \in DB$.

For example, if DB is the set {mother(ann,bob), father(bob,julie), father(bob,chris)}, then the extension of the clause

$$\text{grandmother(X,Y)} \leftarrow \text{mother(X,Z), father(Z,Y)}$$

with respect to DB is the set $DB \cup$ {grandmother(ann,julie), grandmother(ann,chris)}.

As notation, if e, f_1, ..., f_n are facts, DB is a database, and C is a clause, we will also write $DB \vdash e$ if $e \in DB$, $DB \vdash f_1, \ldots, f_n$ if $\forall i (f_i \in DB)$, and $DB \wedge C \vdash f$ if $f \in ext(C, DB)$. This will allow some statements to be made a little more concisely.

Pac-learnability for Clauses

Unlike traditional inductive learning systems, most inductive logic programming learning systems accept two inputs: a set of examples and a database DB. A hypothesis is then constructed of the form $P \wedge DB$, where P is a logic program. The additional input to the learner of DB requires a slight extension to the formal model.

Following previous work [Cohen, 1993a], if LANG is a set of clauses and DB is a logic program, then we use LANG$[DB]$ to denote the set of all pairs of the form (C, DB) such that $C \in$ LANG; each such pair represents the set $ext(C, DB)$. If $\mathcal{DB}$ is a set of databases, then the *family of languages* LANG$[\mathcal{DB}]$ represents the set of all languages LANG$[DB]$ where $DB \in \mathcal{DB}$. We will usually be interested in the set of databases a-$\mathcal{DB}$, defined to be the set of all databases containing facts of arity a or less.

We define a family of languages LANG$[\mathcal{DB}]$ to be *uniformly pac-learnable* iff there is a polynomial-time algorithm PACLEARN$(DB, S^+, S^-, \epsilon, \delta)$ such that for every $DB \in \mathcal{DB}$, PACLEARN$_{DB}(S^+, S^-, \epsilon, \delta)$ is a pac-learning algorithm for LANG$[DB]$, where PACLEARN$_{DB}$ is simply PACLEARN with its first argument fixed to DB. Showing uniform pac-learnability is similar to showing that $\forall DB \in \mathcal{DB}$ LANG$[DB]$ is pac-learnable, but stronger. The additional requirement is that there be a *single* algorithm that works for all databases $DB \in \mathcal{DB}$, and that this algorithm runs in time polynomial in the size of DB.

As to the other details of the learning model, we define the size complexity of a database DB to be its cardinality, which will usually be denoted n_b, and the size complexity of a clause $A \leftarrow B_1, \ldots, B_l$ to be l. Examples will be facts, and the size of an example is its arity; thus, we assume the head of the target clause will have large arity, and the literals in the body will have small arity.

Constant-depth Determinate Clauses

Muggleton and Feng [1992] have introduced a pair of useful restrictions on clauses called *determinacy* and *depth*. Following Muggleton and Feng, if $A \leftarrow B_1, \ldots, B_r$ is an (ordered) Horn clause, then the *input variables* of the literal B_i are those variables appearing in B_i which also appear in the clause $A \leftarrow B_1, \ldots, B_{i-1}$, and all other variables appearing in B_i are *output variables*. A literal B_i is *determinate* (with respect to DB and X) if for every possible substitution σ that unifies A with some $e \in X$ such that $DB \vdash (B_1, \ldots, B_{i-1})\sigma$ there is at most one substitution θ so that $DB \vdash B_i\sigma\theta$. A clause is determinate if all of its literals are determinate. Also define the *depth* of a variable appearing in a clause $A \leftarrow B_1, \ldots, B_r$ as follows. Variables appearing in the head of a clause have depth zero. Otherwise, let B_i be the first literal containing the variable V, and let d be the maximal depth of the input variables of B_i (or zero, if B_i has

no input variables); then the depth of V is $d+1$. The depth of a clause is the maximal depth of any variable in the clause.

Informally, a literal is determinate if its output variables have only one possible binding, given DB and the binding of the input variables. The depth of a variable X is the number of previous variables Y on which the binding of X depends. Muggleton and Feng use the term *ij-determinate* to describe the set of determinate clauses of depth i or less over a background theory DB containing atoms of arity j or less. As noted above, ij-determinate clauses are known to be pac-learnable, and are used in a number of practical learning systems.

The Locality Constraint

Although determinacy is often useful, there are many practical learning problems for which determinate clauses alone are not sufficient [Cohen, 1993c]. We will now consider an alternative restriction on clauses.

Let the *free variables* of a clause be those variables that appear in the body of the clause but not in the head, and let V_1 and V_2 be two free variables appearing in a clause $A \leftarrow B_1, \ldots, B_r$. We will say that V_1 *touches* V_2 if they appear in the same literal, and that V_1 *influences* V_2 if it either touches V_2, or if it touches some variables V' that influences V_2. (Thus *influences* and *touches* are both symmetric and reflexive relations, and *influences* is the transitive closure of *touches*.) The *locale* of a variable V is the set of literals $\{B_{i_1}, \ldots, B_{i_l}\}$ that contain either V, or some variable influenced by V.

Informally, variable V_1 influences variable V_2 if, for a ground query, the choice of a binding for V_1 can affect the possible choices of bindings for V_2. The following examples illustrate these terms: free variables are in boldface type, and the locale of each free variable is underlined.

father(F,S) ←
    son(S,F), husband(F,**W**).
no_payment_due(S) ←
    enlist(S,**PC**),peace_corps(**PC**).
draftable(S) ←
    citizen(S,**C**),united_states(**C**),
    age(S,**A**),(**A**≥18),(**A**≤26).

Notice that the influence relation applies only to free variables. Thus in the third clause above, the variable S is *not* influenced by C, and hence *age(S,A)* is not in the locale of C.

Finally, let the *locality* of a clause be the cardinality of the largest locale of any free variable in that clause. (For instance, the clauses in the example above have locality one, two and three respectively.) Clauses with no free variables are defined to have locality zero. We use k-LOCAL to denote the set of clauses with locality k or less.

Formal Results

Learnability of k-local Clauses

Our first formal result is the following.

Theorem 1 *For any fixed k and a, the language family k-LOCAL[a-$\mathcal{DB}$] is uniformly pac-learnable.*

Proof: As there are only n_e variables in the head of the clause, and every new literal in the body can introduce at most a new variables, any size k locale can contain at most $n_e + ak$ distinct variables. Also note that there are at most n_b distinct predicates in the database DB. Since each literal in a locality has one predicate symbol and at most a arguments, each of which is one of the $n_e + ak$ variables, there are only $n_b(n_e + ak)^a$ different literals that could appear in a locality, and hence at most $p = (n_b(n_e + ak)^a)^k$ different[1] localities of length k. Let us denote these localities as $LOC_1, \ldots, LOC_p$. Note that for constant a and k, p is polynomial in n_e and n_b.

Now, notice that every clause of locality k can be written in the form $C = A \leftarrow LOC_{i_1}, \ldots, LOC_{i_r}$ where each LOC_{i_j} is one of the possible locales, and no free variable appears in more than one of the LOC_{i_j}'s. Since no free variables are shared between locales, the different locales do not interact, and hence $e \in ext(C, DB)$ exactly when $e \in ext(A \leftarrow LOC_{i_1}, DB)$, and $\ldots e \in ext(A \leftarrow LOC_{i_1}, DB)$. In other words, C can be decomposed into a conjunction of components of the form $A \leftarrow LOC_{i_j}$. One can thus use Valiant's [1984] technique for monomials to learn C.

In a bit more detail, the following algorithm will pac-learn k-local clauses. We will assume without loss of generality that DB contains an equality predicate.[2] The learner initially hypothesizes the most specific k-local clause, namely

$$A \leftarrow LOC_1, \ldots, LOC_p$$

(The predicate and arity of the head A can be determined from any of the positive examples, and because DB contains an equality predicate, one can assume that all of the variables in A are distinct.) The learner then examines each positive example e in turn, and deletes from its hypothesis all LOC_i such that $e \notin ext(A \leftarrow LOC_i, DB)$. (Note that e is in this extension exactly when $\exists\theta : DB \vdash LOC_i\sigma_e\theta$ where σ_e is the most general substitution such that $A\sigma_e = e$. To see that this condition can be checked in polynomial time, recall that θ can contain at most ak free variables, and DB can contain at most an_b constants; hence at most $(an_b)^{ak}$ substitutions θ need be checked, which is polynomial.) Following the argument used for Valiant's procedure, this algorithm will pac-learn the target concept. ■

[1] Up to renaming of variables.

[2] That is, that there is a predicate symbol *equal* such that $equal(t_1, t_2) \in DB$ for every pair of constant symbols t_1, t_2 appearing in DB.

We note that the theorem actually proves a stronger result than stated, as it also shows learnability with one-sided error in an on-line model.

The Expressive Power of k-local Clauses

The importance of Theorem 1 depends on the usefulness of k-local clauses as a representation language. While there is some experimental evidence that suggests highly local clauses are, in fact, useful on natural problems [Cohen, 1993c], the principle reason for believing them to be useful is the following result.

Theorem 2 *For every $DB \in a\text{-}\mathcal{DB}$ and every depth-d determinate clause C, there is clause $C' \in k$-LOCAL$[DB]$ such that $ext(C', DB) = ext(C, DB)$, and the size of C' is no greater than k times the size of C, where $k = a^{d+1}$.*

In other words, although a constant-depth determinate clause may have unbounded locality, for every constant-depth determinate clause there is a *semantically equivalent* constant-locality clause. Moreover, this clause is of comparable size and uses precisely the same database predicates. However, the converse is *not* true: for example, the 1-local clause *parent(P)←child(P,C)* has no determinate equivalent.[3] Thus the language of constant-locality clauses is strictly more expressive than the language of constant-depth determinate clauses.

(An attentive reader might be concerned about the locality "constant" $k = a^{d+1}$, which is exponential in depth; this means that the learning algorithm sketched in Theorem 1, if used to learn determinate clauses, will be doubly exponential in depth. Notice, however, that existing algorithms for determinate clauses [Muggleton and Feng, 1992; Džeroski *et al.*, 1992] are also doubly exponential in depth.)

In the remainder of this section, we will give a rigorous proof of Theorem 2. The basic idea of the proof is illustrated by example in Figure 1.

Proof: We will first introduce some additional notation. If $C = A \leftarrow B_1, \ldots, B_r$ is a clause, literal B_i *directly supports* literal B_j iff some output variable of B_i is an input variable of B_j, and that literal B_i *indirectly supports* B_j iff B_i directly supports B_j, or if B_i directly supports some B_k that indirectly supports B_j. Now, for each B_i in the body of C, let LOC_i be the conjunction

$$LOC_i = B_{j_1}, \ldots, B_{j_{k_i}}, B_i$$

where the B_j's are all of the literals of C that support B_i (either directly or indirectly) appearing in the same order that they appeared in C. Next, let us introduce for $i = 1, \ldots, r$ a substitution

$$\sigma_i = \{Y = Yi : \text{variable } Y \text{ occurs in } LOC_i \text{ but not } A\}$$

[3] Over a DB in which parents may have more than one child and in which no other predicates exist.

We then define $LOC'_i = LOC_i\sigma_i$. The effect of this last step is that $LOC'_1, \ldots, LOC'_r$ are copies of $LOC_1, \ldots, LOC_r$ in which variables have been renamed so that for $i \neq j$ the free variables of LOC'_i are different from the free variables of LOC'_j. Finally, let C' be the clause $A \leftarrow LOC'_1, \ldots LOC'_r$.

An example of this construction is given in Figure 1. We strongly suggest that the reader refer to the example at this point.

For a depth-d clause C, we make the following claims: (a) C' is k-local, for $k = a^{d+1}$, (b) C' is at most k times the length of C and (c) if C is determinate, then C' has the same extension as C. In the remainder of the proof, we will establish these claims.

To establish the first two claims it is sufficient to show that the number of literals in every LOC'_i (or equivalently, every LOC_i) is bounded by k. To establish this, let us define $N(d)$ to be the maximum number of literals in any LOC_i corresponding to a B_i with *input* variables at depth d or less. Clearly for any $DB \in a\text{-}\mathcal{DB}$ and any depth d-determinate clause C over DB the function $N(d)$ is an upper bound on k.

The function $N(d)$ is bounded in turn by the following lemma.

Lemma 3 *For any $DB \in a\text{-}\mathcal{DB}$, $N(d) \leq \sum_{i=0}^{d} a^i$ (and hence $N(d) \leq a^{d+1}$).*

Proof of lemma: By induction on d. For $d = 0$, no literals will support B_i. Thus each locality LOC_i will contain only the literal B_i, and $N(0) = 1$.

Now assume that the lemma holds for $d-1$ and consider a literal B_i with inputs at depth d. If $B_{j_1}, \ldots, B_{j_r}$ are the literals that directly support B_i, then LOC_i can be no larger than $LOC_{j_1}, \ldots, LOC_{j_r}, B_i$. Since any literal B_{j_k} that directly supports B_i must be at depth $d-1$ or less, and since there no more than a input variables of B_i, there are at most a different B_{j_k}'s that directly support B_i. Putting this together, and using the inductive hypothesis that $N(d-1) \leq \sum_{i=1}^{d-1} a^i$, we see that

$$N(d) \leq aN(d-1) + 1 \leq a(\sum_{i=0}^{d-1} a^i) + 1 = \sum_{i=0}^{d} a^i$$

By induction, the lemma holds. ∎

Now we consider the second claim: that for any determinate C, the C' constructed above has the same extension. The first direction of this equivalence actually holds for any clause C:

Lemma 4 *If $f \in ext(D, DB)$, then $f \in ext(C', DB)$.*

Proof of lemma: Consider the substitutions σ_i introduced in the construction of C'. Since each free variable in LOC_i is given a distinct name in LOC'_i, σ_i is a one-to-one mapping, and since the free variables in the LOC'_i's are distinct, the substitution $\sigma = \bigcup_{i=1}^{r} \sigma_i^{-1}$

The starting point: Below is a depth-2 determinate clause.

```
good_grant_proposal(X) ←
    author(X,PI),          % B1
    employer(PI,U),        % B2, supported by B1
    prestigious(U),        % B3, supported directly by B2 and indirectly by B1
    topic(X,T),            % B4
    trendy(T).             % B5, supported by B4
```

Step 1: for each B_i build a locality LOC_i containing all supporting B_j's.

LOC_1 = author(X,PI)
LOC_2 = author(X,PI),employer(PI,U)
LOC_3 = author(X,PI),employer(PI,U),prestigious(U)
LOC_4 = topic(X,T)
LOC_5 = topic(X,T),trendy(T)

Step 2: Rename variables so that all free variables appear in only a single locale.

LOC'_1 = author(X,PI1)
LOC'_2 = author(X,PI2),employer(PI2,U2)
LOC'_3 = author(X,PI3),employer(PI3,U3),prestigious(U3)
LOC'_4 = topic(X,T4)
LOC'_5 = topic(X,T5),trendy(T5)

Step 4: Collect the LOC'_i's into a single clause.

```
good_grant_proposal(X) ←
    author(X,PI1),
    author(X,PI2),employer(PI2,U2),
    author(X,PI3),employer(PI3,U3),prestigious(U3),
    topic(X,T4),
    topic(X,T5),trendy(T5).
```

Figure 1: Constructing a local clause equivalent to a determinate clause

is well-defined. As an example, for the clause C' from Figure 1, we would have

$$\sigma = \{ \begin{array}{lll} \text{PI1} = \text{PI}, & \text{PI2} = \text{PI}, & \text{PI3} = \text{PI}, \\ \text{U2} = \text{U}, & \text{U3} = \text{U}, & \\ \text{T4} = \text{T}, & \text{T5} = \text{T} \ \} & \end{array}$$

Applying this substitution to C' will simply "undo" the effect of renaming the variables, so that

$$C'\sigma = (A \leftarrow LOC_1, \ldots, LOC_r)$$

Now, assume $f \in ext(C, DB)$. Then there is by definition some substitution θ so that all literals in the body of the clause $C\theta$ are in DB. Since $C'\sigma$ contains the same set of literals in its body as C, clearly for the substitution $\theta' = \sigma \circ \theta$ all literals in the body of the clause $C'\theta'$ are in DB, and hence $f \in ext(C', DB)$. ■

We must finally establish the converse of Lemma 4. This direction of the equivalence requires that C be determinate.

Lemma 5 *If a fact $f \in ext(C', DB)$ and C is determinate, then $f \in ext(C, DB)$.*

Proof of lemma: If $f \in ext(C', DB)$, then either $f \in DB$, in which case the lemma holds trivially, or there must be some θ' that is a "witness" that $f \in ext(C', DB)$—by which we mean a θ' such that $A'\theta = f$, and for every B'_i from the body of C', $B'_i\theta \in DB$. Define a variable Yi in C' to be a "copy" of $Y \in C$ if Yi is a renaming of Y—*i.e.*, if $Yi\sigma_i^{-1} = Y$ for the σ_i defined in the lemma above. Certainly if C is determinate then C' is determinate; further, for a determinate C', θ' must map every copy of Y to the same constant t_Y. (This can be proved by picking two copies Yi and Yj of Y and then using induction over the depth of Y to show that they must be bound to the same constant.) Hence, let us define the substitution

$$\theta = \{Y = t_Y : \text{copies of } Y \text{ in } C' \text{ are bound to } t_Y \text{ by } \theta'\}$$

Clearly, for all $i : 1 \leq i \leq r$, $LOC_i\theta = LOC_i\theta'$; hence if θ' witnesses that $f \in ext(C', DB)$ then θ witnesses that $f \in ext(C, DB)$. ■

We have now established that C' is k-local, of bounded size, and is equivalent to C. This concludes the proof of the theorem. ■

Conclusions

This paper continues a line of research which is intended to broaden the theoretical foundations of inductive logic programming systems by formally investigating the learnability of restricted logic programs. Previous work has shown that nonrecursive constant-depth determinate Datalog clauses are pac-learnable [Džeroski *et al.*, 1992]. More recently, it was shown that relaxing any of these conditions leads to a language that is hard to learn; in particular, the languages of recursive constant-depth determinate clauses, nonrecursive log-depth determinate clauses, and nonrecursive constant-depth indeterminate clauses are all hard to pac-learn [Kietz, 1993; Cohen, 1993a]. Furthermore, relaxing the condition of determinacy has proven especially difficult.

This paper has proposed a new restriction on indeterminate free variables called *locality*. Informally, a clause is k-local if the binding picked for any free variable affects the success or failure of only k literals. Every clause of locality k must have depth k or less, and hence this language is more restrictive than the language of constant-depth nonrecursive clauses. However, the k-local restriction is incomparable to other restrictions on indeterminate clauses previously considered in the literature, notably restricting the number of free variables in a clause [Haussler, 1989; Cohen, 1993a].

In this paper, we have shown that the language of k-local clauses is pac-learnable, and also that the language is strictly more expressive that the language of constant-depth determinate clauses. Hence it is a pac-learnable generalization of the language of constant-depth determinate clauses. The existence of such a generalization is somewhat surprising, given the negative results of Cohen [1993a] and Kietz [1993]. Note, however, that Cohen and Kietz considered only *syntactic* generalizations of constant-depth determinacy, and the language of k-local clauses is a semantic (but not syntactic) generalization.

A number of further topics are suggested by these results. While the positive pac-learnability result is encouraging, the algorithm sketched in Theorem 1 seems relatively inefficient. It remains to be seen whether learning algorithms for k-local clauses that are both well-understood and practically useful can be designed. Also, it would be interesting to explore the learnability of recursive k-local clauses. Since the hardness results for recursive constant-depth determinate clauses of Cohen [1993a] are "representation-independent", it is immediate that k-local clauses with arbitrary recursion are not pac-learnable; however, the learnability of the linear recursive case (for example) is open.

References

Cohen, William W. 1992. Compiling knowledge into an explicit bias. In *Proceedings of the Ninth International Conference on Machine Learning*, Aberdeen, Scotland. Morgan Kaufmann.

Cohen, William W. 1993a. Cryptographic limitations on learning one-clause logic programs. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, Washington, D.C.

Cohen, William W. 1993b. A pac-learning algorithm for a restricted class of recursive logic programs. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, Washington, D.C.

Cohen, William W. 1993c. Rapid prototyping of ILP systems using explicit bias. In *Proceedings of the 1993 IJCAI Workshop on Inductive Logic Programming*, Chambery, France.

Džeroski, Sašo; Muggleton, Stephen; and Russell, Stuart 1992. Pac-learnability of determinate logic programs. In *Proceedings of the 1992 Workshop on Computational Learning Theory*, Pittsburgh, Pennsylvania.

Haussler, David 1989. Learning conjunctive concepts in structural domains. *Machine Learning* 4(1).

Kietz, Jorg-Uwe 1993. Some computational lower bounds for the computational complexity of inductive logic programming. In *Proceedings of the 1993 European Conference on Machine Learning*, Vienna, Austria.

Lavrač, Nada and Džeroski, Sašo 1992. Background knowledge and declarative bias in inductive concept learning. In Jantke, K. P., editor 1992, *Analogical and Inductive Inference: International Workshop AII'92*. Springer Verlag, Daghstuhl Castle, Germany. Lectures in Artificial Intelligence Series #642.

Lloyd, J. W. 1987. *Foundations of Logic Programming: Second Edition*. Springer-Verlag.

Muggleton, Stephen and Feng, Cao 1992. Efficient induction of logic programs. In *Inductive Logic Programming*. Academic Press.

Quinlan, J. Ross 1990. Learning logical definitions from relations. *Machine Learning* 5(3).

Quinlan, J. Ross 1991. Determinate literals in inductive logic programming. In *Proceedings of the Eighth International Workshop on Machine Learning*, Ithaca, New York. Morgan Kaufmann.

Valiant, L. G. 1984. A theory of the learnable. *Communications of the ACM* 27(11).