

# Version Spaces Without Boundary Sets

**Haym Hirsh**  
hirsh@cs.rutgers.edu  
Computer Science Department  
Rutgers University  
New Brunswick, NJ 08903

**Nina Mishra**  
nmishra@uiuc.edu  
Computer Science Department  
University of Illinois at Urbana  
Urbana, IL 61801

**Leonard Pitt**  
pitt@cs.uiuc.edu  
Computer Science Department  
University of Illinois at Urbana  
Urbana, IL 61801

## Abstract

This paper shows that it is not necessary to maintain boundary sets to reason using version spaces. Rather, most of the operations typically performed on version spaces for a concept class can be tractably executed directly on the training data, as long as it is tractable to solve the consistency problem for that concept class — to determine whether there exists any concept in the concept class that correctly classifies the data. The equivalence of version-space learning to the consistency problem bridges a gap between empirical and theoretical approaches to machine learning, since the consistency problem is already known to be critical to learning in the PAC (Probably Approximately Correct) sense. By exhibiting this link to the consistency problem, we broaden the class of problems to which version spaces can be applied to include concept classes where boundary sets can have exponential or infinite size and cases where boundary sets are not even well defined.

## 1 Introduction

The problem of *inductive learning* is to extrapolate from a collection of *training data* — a set of examples, each labeled as either positive or negative by some unknown target concept — a concept definition that accurately labels future, unlabeled data. Most inductive learning algorithms operate over some *concept class* — the set of concepts that the learner can potentially generate over all possible sets of training data. Mitchell's (1982) introduction of the notion of a *version space* provided a useful conceptual tool for inductive learning. Given a concept class  $C$ , the version space for a set of examples is simply the set of concepts in  $C$  that are consistent with the data. To represent a version space, Mitchell proposed maintaining only the set of maximally general ( $G$ ) and maximally specific ( $S$ ) consistent concepts, called *boundary sets*, since they bound the set of all concepts in the version space. However, although it is more efficient than explicitly maintaining all elements of the version space, this representation has a number of serious limitations. For example,

Haussler (1988) showed that even for the concept class of positive-monotone monomials over  $n$  variables (i.e., simple conjunctions of subsets of  $n$  unnegated Boolean attributes) there are cases where after a number of examples linear in  $n$ , the boundary set  $G$  has size exponential in  $n$ . Further, for infinite concept classes, boundary sets can have potentially infinite size, and in some cases (for *inadmissible* concept classes (Mitchell 1982)) it is not even possible to represent a version space by its boundary sets. Section 5 presents version spaces exhibiting such pathologies.<sup>1</sup>

A number of papers have proposed alternative representations for version spaces to overcome some of these limitations. Particularly relevant to this work, VanLehn and Ball (1987) proposed maintaining an approximate and overgeneral  $G$  set plus the set  $N$  of all negative examples (which need not any longer be reflected in the new  $G$  set) as a way of tractably but approximately learning restricted classes of context-free grammars. (Interestingly, VanLehn and Ball also totally do away with the  $S$  boundary set, maintaining solely the set  $P$  of positive examples instead. However, this was only due to the fact that, for the languages they learn, the  $S$  set is the trivial concept that classifies only the given positive examples as positive and all else negative, and thus  $P$  and  $S$  are trivially equivalent.) Smith and Rosenbloom (1990) showed that a similar idea — maintaining the  $S$  and  $G$  boundary sets for all the positive data plus only a selected subset of the negative data, along with the set  $N$  of unprocessed negative examples — was sufficient to guarantee tractable version-space learning for a constrained class of conjunctive languages. Subsequent work (Hirsh 1992) took this further, doing away with the set  $G$  altogether, and instead using an  $[S, N]$  representation that maintains only the boundary set  $S$  together with the set  $N$  of all negative examples, and showing that for many common concept classes, most of what could be done with

---

<sup>1</sup>Version spaces also face difficulties in their limited capacity to handle noisy data or concept classes that do not contain the target concept (Utgoff 1986; Hirsh 1994), but we concern ourselves in this paper solely with the representational difficulties faced even when these are not at issue.

the  $[S, G]$  representation could be accomplished with the  $[S, N]$  representation, only with tractability guarantees due to its more modest representational requirement of maintaining a single (non-exponentially-large) boundary set.

Unfortunately, there are also cases where *both* boundary sets can be prohibitively large. Hausler (1989) gives one such example, for conjunctions of existentially quantified predicates, and in Section 5 we show that a similar situation can occur for the propositional language of 1-decision lists (1-DLs). The  $[S, N]$  representation is inadequate in these cases, in that it still must maintain one of the two exponentially large boundary sets.

In this paper we show that focusing on the representation of a version space — understanding the behavior of boundary sets, exploring other version-space representations, etc. — is not the key issue for effective version-space learning for a concept class  $C$ . Rather, the focus should be on finding efficient algorithms for the *consistency problem* for  $C$  — the problem of determining if *any* concept in  $C$  exists that correctly classifies a given set of positive and negative examples. We demonstrate that almost all operations performed on version spaces for any concept class  $C$  can be efficiently executed if and only if there is an efficient solution to the consistency problem for  $C$ . In version-space terminology, almost all version-space operations can be efficiently executed if and only if there is an efficient way of determining whether a version space has *collapsed*, i.e., is empty.

Our “representation” of a version space is thus not much of a representation at all — we simply store the positive ( $P$ ) and negative ( $N$ ) examples, and tractably perform version-space operations directly on these two sets.<sup>2</sup> Note that it is impossible to show that the  $[P, N]$  representation can grow faster than its input because the input *is* its representation. Thus infinite and inadmissible concept classes do not pose a problem for this representation. The question simply becomes one concerning the tractability of *executing* version-space operations on this representation. For those concept classes where operations on the standard boundary-set representation are provably tractable (i.e., can be executed in time polynomial in relevant problem parameters), the  $[P, N]$  representation is also guaranteed to be tractable, even if only by first computing the boundary sets. In the other direction, however, there are cases (such as 1-decision lists) where the  $[P, N]$  representation is tractable (because there is an efficient solution to the consistency problem) and, yet, the boundary-set representation is not.

It is particularly nice that the tractability of the consistency problem plays a key role for version-space learning, given the importance it plays in the computational learning theory literature and the study this

<sup>2</sup>Thus our results can be viewed as providing a lazy learning algorithm for version spaces (Aha 1997).

problem has received there. For example, if there exists an algorithm for a concept class  $C$  that outputs a consistent hypothesis of “small” size, then  $C$  is PAC learnable (Blumer *et al.* 1987). As another example, if the VC-dimension of a concept class  $C$  grows polynomially and if there exists an algorithm that outputs a consistent hypothesis from  $C$ , then  $C$  is PAC-learnable (Blumer *et al.* 1989). A partial converse is also known (Pitt & Valiant 1988; Blumer *et al.* 1989).

One of the main insights and sources of power concerning version spaces is that they provide a way to classify future, unseen examples even when insufficient data have been obtained to identify the unique identity of the target concept. We show with a simple argument that doing such classification of future examples is exactly as hard as solving the consistency problem. The way the consistency problem can be used to classify an example  $x$  according to a version space induced by positives  $P$  and negatives  $N$  is actually quite simple. We run the consistency algorithm twice. The first time, we run the algorithm assuming  $x$  is a positive example (i.e., with positives  $P \cup \{x\}$  and negatives  $N$ ). If no consistent concept exists (alternatively, if the version space collapses), then every consistent concept must classify  $x$  negative. Otherwise, we run the consistency algorithm assuming  $x$  is a negative example (i.e., with positives  $P$  and negatives  $N \cup \{x\}$ ). Similarly, if no consistent concept exists, then every consistent concept must classify  $x$  positive. If neither case holds, there must be at least one consistent concept that classifies  $x$  positive and at least one that classifies  $x$  negative — the label of  $x$  is not determined by the version space, and, consequently,  $x$  is labeled “?”.

We begin by proving that classifying an example in a manner consistent with a version space is exactly the consistency problem (Section 3). Next, we show that almost all of the standard operations on version spaces can be performed using an efficient solution to the consistency problem (Section 4). We conclude with applications of the main result (Section 5).

## 2 Definitions and Notation

A *concept*  $c$  is a subset  $c \subseteq X$  of a space  $X$  of *examples*. An example  $x \in X$  is said to be labeled *positive* by  $c$  if  $x \in c$ , and *negative* otherwise. A *concept class*,  $C \subseteq 2^X$ , is a collection of concepts. For a set of positive examples,  $P$ , negative examples,  $N$ , and a concept class,  $C$ , the *version space* is the set of all concepts in  $C$  consistent with  $P$  and  $N$  (Mitchell 1982). We use  $C_{P,N}$  to denote the version space induced by  $P$  and  $N$ . Specifically,  $C_{P,N} = \{c \in C : P \subseteq c, \text{ and } N \cap c = \emptyset\}$ . If  $C_{P,N}$  is empty we say that the version space has *collapsed*; if it contains a single concept we say that the version space has *converged*.

If every element of a version space labels an example the same way, then that example can be unambiguously labeled even if the version space contains multiple

concepts, since no matter which one was the “correct” one, they all agree on how to label the given example. A version space  $C_{P,N}$  can therefore be viewed as inducing a function,  $\text{classify}(C_{P,N})$ , that, given an example, outputs a label that reflects the consensus of concepts in  $C_{P,N}$ :

$$\text{classify}(C_{P,N})(x) = \begin{cases} \text{“}\emptyset\text{”} & \text{if } C_{P,N} \text{ is empty} \\ \text{“}+\text{”} & \text{if } x \in c \text{ for all } c \in C_{P,N} \\ \text{“}-\text{”} & \text{if } x \notin c \text{ for all } c \in C_{P,N} \\ \text{“}?\text{”} & \text{otherwise} \end{cases}$$

We sometimes abuse notation and use  $C_{P,N}(x)$  instead.

The tractable computation of the classify function is one of the key questions that we study here:

**Definition 2.1** *A concept class  $C$  is (efficiently) version-space predictable if there exists an algorithm that, given a set of positive examples,  $P$ , a set of negative examples,  $N$ , and an example,  $x$ , outputs  $C_{P,N}(x)$  in time polynomial in  $|P|$ ,  $|N|$ , and  $|x|$ .*

Note that we are really only interested in concept classes  $C$  that are “missing” some concepts, i.e.,  $C \neq 2^X$ . Otherwise, if  $C = 2^X$  (e.g., if  $C$  is the class of DNF formulas) then it is easy to see that  $C$  is trivially version-space predictable.

Finally, the consistency problem for  $C$  can be summarized as the problem of determining if there is a concept in  $C$  that correctly labels a given set of positive and negative examples. More formally stated,

**Definition 2.2** *The consistency problem for  $C$  is: Given a set of positive examples,  $P$ , and negative examples,  $N$ , is  $C_{P,N} \neq \emptyset$ ?<sup>3</sup>*

The above definition is equivalent to: Has the version space induced by  $P$  and  $N$  collapsed? We’ll say that the consistency problem for  $C$  is *efficiently* computable if there is an algorithm for the consistency problem for  $C$  that runs in time polynomial in  $|P|$  and  $|N|$ .

### 3 Consistency and Version-Space Predictability

A key result of this paper is that version-space predictability is equivalent to the consistency problem. We begin with a simple proposition.

**Proposition 3.1** *For all  $x$  in  $X$ ,*

1.  $(C_{P \cup \{x\}, N} = \emptyset) \Leftrightarrow (x \notin c \text{ for every } c \in C_{P,N})$
2.  $(C_{P, N \cup \{x\}} = \emptyset) \Leftrightarrow (x \in c \text{ for every } c \in C_{P,N})$

**Proof:** We show part 1, the proof of part 2 is analogous. For arbitrary  $x$ , if  $x \notin c$  for every  $c \in C_{P,N}$ , then clearly  $C_{P \cup \{x\}, N} = \emptyset$ . Conversely, if  $x \in c$

<sup>3</sup>In many references (Blumer *et al.* 1989; Pitt & Valiant 1988; Aizenstein *et al.* 1997), the consistency-problem definition requires that a consistent hypothesis be explicitly output if one exists. While our main results do not require that a hypothesis be output, all of the algorithms we exhibit in Section 5 actually output consistent hypotheses.

for some concept  $c \in C_{P,N}$ , then  $c \in C_{P \cup \{x\}, N}$ , and  $C_{P \cup \{x\}, N} \neq \emptyset$ .  $\square$

We can use an algorithm for the consistency problem for  $C$  to predict the label of  $x$  according to the version space induced by  $[P, N]$  by simply running the consistency algorithm twice — once with  $x$  in  $P$  and another time with  $x$  in  $N$ . If either is empty, the preceding proposition gives the basis for classifying  $x$ .

**Theorem 3.1**  *$C$  is efficiently version-space predictable  $\Leftrightarrow$  the consistency problem for  $C$  is efficiently computable.*

**Proof:**  $\square \Rightarrow$  Note that  $C_{P,N}$  is empty iff both  $C_{P \cup \{x\}, N}$  and  $C_{P, N \cup \{x\}}$  are empty. Assume  $C_{P,N} \neq \emptyset$ . By definition,  $C_{P,N}(x) = \text{“}+\text{”}$  if and only if  $x \in c$  for all  $c \in C_{P,N}$ . By the second part of Proposition 3.1, this holds if and only if  $C_{P, N \cup \{x\}}$  is empty. Similarly, by definition,  $C_{P,N}(x) = \text{“}-\text{”}$  if and only if  $x \notin c$  for all  $c \in C_{P,N}$ , which, by Proposition 3.1 part 1, holds if and only if  $C_{P \cup \{x\}, N}$  is empty. And, if neither hold, then, the classification is by definition “?”.

It now follows immediately that an algorithm for the consistency problem can be used to solve the version space classification problem: run the consistency algorithm on inputs  $P \cup \{x\}$ ,  $N$  and then again on  $P$ ,  $N \cup \{x\}$ . If only the first one succeeds then  $x$  must be in every concept in the version space, and the version-space algorithm should predict “+”. (Dually if only the second one succeeds.) If both succeed, then  $x$  must be in some concepts in the version space and not in others, and thus, the version space algorithm should predict “?”. If both fail, then the algorithm should predict “?”.

Observe that if the consistency algorithm is efficient, i.e., runs in time  $p(|P|, |N|)$ , where  $p$  is some polynomial, then the version-space prediction algorithm runs in time  $p(|P \cup \{x\}|, |N|) + p(|P|, |N \cup \{x\}|)$ , and is also efficient.

$\Rightarrow$  If  $C$  is version-space predictable then, by definition, the consistency problem is efficiently computable. To determine if there is a concept in  $C$  consistent with  $[P, N]$ , use the version-space prediction algorithm to classify  $C_{P,N}(x)$  for an arbitrary example  $x$  in  $X$ . There is no concept consistent with  $[P, N]$  if and only if the version-space prediction algorithm outputs “?” , i.e., if the version space has collapsed.  $\square$

### 4 Other Version-Space Operations

We now discuss the tractability of other version-space operations using the  $[P, N]$  representation.

**Collapse:** A version space has collapsed iff there is no consistent concept, and thus version-space collapse is trivially testable with an efficient consistency algorithm.

**Concept Membership:** Given a version space  $C_{P,N}$ , one may want to determine if a given concept

$c$  is in  $C_{P,N}$ . The tractability of this operation using the  $[P, N]$  representation only relies on the tractability of labeling an example with a concept:  $c$  is in the version space  $C_{P,N}$  if and only if  $p \in c$  and  $n \notin c$ , for all  $p \in P$  and  $n \in N$ . The tractability of this operation under the  $[P, N]$  representation is independent of the tractability of the consistency problem and the equivalent concept-class properties discussed in the previous section. As long as it is tractable to classify an example with a concept, it is tractable to test whether a concept is in a version space under the  $[P, N]$  representation. This is an interesting contrast to the requirements necessary for the tractability of this operation for the boundary-set representation — that it be tractable to determine whether  $c_1 \subseteq c_2$  for all  $c_1, c_2 \in C$ .

**Update and Retraction:** If generating the representation of a version space for a collection of data requires significant computation one must consider the tractability of updating this representation as new data are obtained. For boundary sets, this requires generating new boundary sets that reflect all past data (summarized by the current boundary sets) plus the new data. For the  $[P, N]$  representation this update operation is trivial: add the example to  $P$  if the example is positive, and add it to  $N$  if it is negative.

Retracting data is likewise trivial: simply remove the example from the set in which it occurs,  $P$  or  $N$ . This operation is not often considered with version spaces primarily due to the fact that it is difficult to retract an example once it is reflected in the current version-space representation (although retraction is important in handling noisy data with version spaces (Idestam-Almqvist 1989)).

**Intersection:** Previous work (Hirsh 1994) showed that version-space intersection could form the basis for an alternative approach to learning with version spaces. The intersection of version spaces  $C_{P_1, N_1}$  and  $C_{P_2, N_2}$ , denoted  $C_{P_1, N_1} \cap C_{P_2, N_2}$ , is the set of concepts that contain all the examples in both  $P_1$  and  $P_2$  and do not contain any examples in  $N_1$  and  $N_2$ . The question becomes whether one can efficiently compute the representation of the version space for  $C_{P_1, N_1} \cap C_{P_2, N_2} = C_{P_1 \cup P_2, N_1 \cup N_2}$  given the representations for  $C_{P_1, N_1}$  and  $C_{P_2, N_2}$ . As with the preceding Update operation, this is trivial — the  $P$  set for the new version space is just  $P_1 \cup P_2$  and the new  $N$  set is just  $N_1 \cup N_2$ .

**Subset Testability and Equality:** Version spaces are simply sets, and thus one can ask whether the version space for one set of data is a subset of the version space for a second set of data. At first this may appear to be a problem that is harder than the consistency problem — for example, one version space may be a subset of a second even if they are based on disjoint sets of training data. Initially it may seem that it would be necessary to enumerate the elements in both version spaces, or at least their boundary sets, to do

subset-testing. However, not only is subset testing no harder than the consistency problem, the two problems are equivalent, as we now show.

**Definition 4.1** A concept class  $C$  is subset-testable if there exists an algorithm that, given two sets of positive examples,  $P_1$  and  $P_2$ , and two sets of negative examples,  $N_1$  and  $N_2$ , determines in time polynomial in  $|P_1|, |P_2|, |N_1|$ , and  $|N_2|$ , whether  $C_{P_1, N_1} \subseteq C_{P_2, N_2}$ .

To show that subset-testability is equivalent to the efficient computation of the consistency problem, we show how a tractable solution to either efficiently solves the other.

**Theorem 4.1**  $C$  is subset-testable if and only if the consistency problem for  $C$  is efficiently computable.

**Proof:** Note that if  $C_{P_1, N_1} \subseteq C_{P_2, N_2}$ , every element  $p \in P_2$  must be classified as “+” by  $C_{P_1, N_1}$  and every element  $n \in N_2$  must be classified as “-” by  $C_{P_1, N_1}$ . This can be tested using an efficient procedure for the consistency problem by returning true if  $C_{P_1, N_1 \cup \{p\}} = \emptyset$  for each  $p \in P_2$  and  $C_{P_1 \cup \{n\}, N_1} = \emptyset$  for each  $n \in N_2$ , and otherwise return false.

To use an efficient test for  $C_{P_1, N_1} \subseteq C_{P_2, N_2}$  as the basis for efficiently solving the consistency problem, note that  $C_{P,N} = \emptyset$  if and only if  $C_{P,N} \subseteq C_{\{x\}, \{x\}}$  for any  $x \in X$ .  $\square$

To test Equality, observe that two version spaces  $C_{P_1, N_1}$  and  $C_{P_2, N_2}$  are equal if and only if both  $C_{P_1, N_1} \subseteq C_{P_2, N_2}$  and  $C_{P_2, N_2} \subseteq C_{P_1, N_1}$  — which by Theorem 4.1 can be efficiently determined when the consistency problem for  $C$  admits an efficient algorithm.

**Union and Difference:** Since version spaces are simply sets, in addition to intersection one can consider the union and set difference of two sets. Unfortunately, although these two operations can be useful in learning with version spaces (Hirsh 1994), version spaces are not closed under either union or set difference — there may be no set of examples that gives a version space that is equal to the union or difference of two given version spaces. Thus, we do not offer here a way to compute the union or difference of version spaces (whereas in some cases the boundary-set representation can still apply (Gunter *et al.* 1991; Hirsh 1991)).

**Convergence:** A version space  $C_{P,N}$  is said to have converged if there is exactly one concept in  $C$  consistent with  $P$  and  $N$ , i.e., if  $|C_{P,N}| = 1$ . Observe that the existence of an efficient algorithm for the consistency problem for  $C$  does *not* necessarily imply the existence of an efficient algorithm for the convergence problem for  $C$ . Intuitively, it is “harder” to determine if there is *exactly* one consistent concept (i.e., convergence) than determine if there is *any* consistent concept.

To instantiate this intuition, we note that for the class  $C$  of monotone formulas, the consistency problem

is efficiently computable. (Simply check if any positive example falls “below”, in the Boolean hypercube, a negative, and vice versa.) However, it is possible to show that the convergence problem is equivalent to determining if a monotone DNF formula is equivalent to a monotone CNF formula. While there are efficient solutions to restricted versions of the equivalence of monotone DNF and CNF problem (Eiter & Gottlob 1995; Johnson, Papadimitriou, & Yannakakis 1988; Lawler, Lenstra, & Rinnooy Kan 1980; Mishra & Pitt 1997), the best known algorithm for the general problem runs in superpolynomial time (Fredman & Khachiyan 1996). So while there is a polynomial-time algorithm for the consistency problem for monotone formulas, the convergence problem would appear to be harder.

This apparent hardness does not disturb us greatly, since the convergence operation does not play as important a role in learning with version spaces as it initially appeared to in Mitchell’s work. In particular, convergence usually requires a very large number of examples: the smaller an unconverged version space is, the longer the wait for a random example that can distinguish them (Haussler 1988). Haussler also showed that under the PAC learning criteria, it is not necessary to generate a converged version space, since any element of a version space for some number of randomly chosen examples will do comparably well on future data. It is thus unusual to wait until a version space becomes singleton, as opposed to, say, selecting a random element of the version space for classification purposes (Norton & Hirsh 1992; 1993).

## 5 Example Concept Classes

We now study the tractability of version-space learning for a number of concept classes. In each case we explore the tractability of the consistency problem, as well as, in some cases, the (in)tractability of boundary-set-based version spaces for that class.

**One Boundary Set Large:** Although Haussler (1988) showed that it is possible to have one boundary set grow large when learning monomials, an even more surprising case of this — where our approach to version spaces now makes learning tractable — is for the concept class of propositional Horn sentences (conjunctions of Horn clauses). It is possible to show with a small training sample that the size of one boundary set can grow exponentially large (Aizenstein & Pitt 1995). Nonetheless, the consistency problem for Horn sentences is efficiently computable. We demonstrate how to construct a Horn sentence,  $H$ , consistent with a given  $P$  and  $N$  whenever one exists. The idea is to construct for each negative example  $n$  a set of clauses that falsify  $n$ , and then remove from this set the clauses that also falsify examples in  $P$ . After this removal, the Horn sentence will necessarily be consistent with  $P$ , and, if it is not consistent with  $N$ , then it can be

shown that no Horn sentence exists that can satisfy  $P$  and falsify  $N$ .

For an example  $n$ , let  $ones(n)$  be the conjunction of all 1-bits of  $n$  and  $zeros(n)$  be the set of all 0-bits of  $n$ . By convention,  $False \in zeros(n)$ . If  $n$  is a negative example, then the following is a Horn sentence which excludes  $n$ :  $clauses(n) = \bigwedge_{z \in zeros(n)} (ones(n) \rightarrow z)$  For example, if  $n = 11001$ , then  $clauses(n) = (x_1x_2x_5 \rightarrow x_3) \wedge (x_1x_2x_5 \rightarrow x_4) \wedge (x_1x_2x_5 \rightarrow False)$ .

Consider the Horn sentence  $H$  obtained by conjoining  $clauses(n)$  for each  $n$  in  $N$ , and then removing any “bad” clauses that exclude points in  $P$ :  $H = \bigwedge_{\exists n \in N: l \in clauses(n) \text{ and } P \not\subseteq l} l$ . Clearly if  $H$  is consistent with  $P$  and  $N$  then there is a Horn sentence consistent with  $P$  and  $N$ . If  $H$  is not consistent, then it can be shown that some negative example  $n$  of  $N$  is not classified negative and, further, no Horn clause exists that can simultaneously classify  $n$  negative and all the examples in  $P$  positive.

**Both Boundary Sets Large:** (Haussler 1988) gives an example over the class of simple conjunctions (1-CNF formulas) where  $|G|$  grows exponentially in the size of the training sample. A dual statement can be made for the class of simple disjunctions (1-DNF formulas) — namely, that  $|S|$  can grow large. Further, it is possible to show for the concept class of conjunctions or disjunctions (1-DNF  $\cup$  1-CNF), that *both*  $|G|$  and  $|S|$  can be large. As this concept class is not exactly natural, we investigate a class that has been previously studied by, for example Rivest (1987), that properly includes 1-DNF  $\cup$  1-CNF, namely 1-decision lists (essentially a maximally unbalanced decision tree). Note that since both boundary sets can be large after a small number of examples, one-sided boundary set representations like the  $[S, N]$  and  $[G, P]$  representation (Hirsh 1992) are ineffective for this concept class.

Consider the class of 1-decision lists over the  $4n$  variables  $u_1, v_1, \dots, u_n, v_n, x_1, y_1, \dots, x_n, y_n$ . Let  $\vec{1}$  be the length  $2n$  example in which every bit position is 1 (analogously for  $\vec{0}$ .) Also, let  $\vec{1}_{2i}$  ( $\vec{0}_{2i}$ ) be the length  $2n$  example with bits  $2i$  and  $2i + 1$  set off (on) and the remaining bits sets on (off). If  $P = \{0_{2i} \vec{1} : i = 1, \dots, n\}$  and  $N = \{\vec{0} 1_{2i} : i = 1, \dots, n\}$  then it can be shown that  $|G|$  and  $|S|$  are *both* at least  $(n!2^n)$ . However, although maintaining even one boundary set in this case requires exponential space, the consistency problem for 1-decision lists is efficiently solvable (Rivest 1987).<sup>4</sup>

**Ill-Defined Boundary Sets:** In a continuous domain, e.g., the Euclidean plane, it is a simple exercise to exhibit situations where both  $G$  and  $S$  are infinite, yet the consistency problem is efficiently solvable.

<sup>4</sup>This is in contrast to Haussler’s (1989) example of conjunctions of existentially quantified predicates for which both  $|G|$  and  $|S|$  grow exponentially, yet the consistency problem is NP-complete.

In particular, the reader will enjoy demonstrating this fact when the concept class is the set of open or closed halfspaces over two real-valued variables. Two examples are sufficient to force both  $G$  and  $S$  to be infinite. (Letting the concept class consist solely of open halfspaces, for example, gives a stronger result that the set  $S$  is not even well-defined.) The efficiency of the consistency problem can be seen by using any polynomial-time algorithm for linear programming.

**When Consistency is NP-hard:** By applying Theorem 3.1 in the other direction, we have that if the consistency problem for  $C$  is NP-hard, then  $C$  is not version-space predictable, unless  $P = NP$ . For example, since results of Pitt and Valiant (1988) show that the consistency problem for  $k$ -term DNF formulas is NP-hard, this class is not version-space predictable, unless  $P = NP$ . However (as their work goes on to suggest), we can still use version spaces for this concept class if we use the richer knowledge representation class of  $k$ -CNF formulas since it includes  $k$ -term DNF formulas and there is a tractable solution to the consistency problem for  $k$ -CNF formulas.

## References

- Aha, D. 1997. Special issue on lazy learning. *Artificial Intelligence Review*, To appear 11(1-5).
- Aizenstein, H., and Pitt, L. 1995. On the learnability of disjunctive normal form formulas. *Machine Learning* 19(3):183-208.
- Aizenstein, H.; Hegedus, T.; Hellerstein, L.; and Pitt, L. 1997. Complexity theoretic hardness results for query learning. *Computational Complexity*, To appear.
- Blumer, A.; Ehrenfeucht, A.; Haussler, D.; and Warmuth, M. K. 1987. Occam's razor. *Inform. Proc. Lett.* 24:377-380.
- Blumer, A.; Ehrenfeucht, A.; Haussler, D.; and Warmuth, M. K. 1989. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM* 36(4):929-965.
- Eiter, T., and Gottlob, G. 1995. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing* 24(6):1278-1304.
- Fredman, M. L., and Khachiyan, L. 1996. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms* 21(3):618-628.
- Gunter, C. A.; Ngair, T.-H.; Panangaden, P.; and Subramanian, D. 1991. The common order-theoretic structure of version spaces and ATMS's (extended abstract). In *Proceedings of the National Conference on Artificial Intelligence*, 500-505.
- Haussler, D. 1988. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence* 36:177-221.
- Haussler, D. 1989. Learning conjunctive concepts in structural domains. *Machine Learning* 4(1):7-40.
- Hirsh, H. 1991. Theoretical underpinnings of version spaces. In *Proceedings of the Twelfth Joint International Conference on Artificial Intelligence*, 665-670. San Mateo, CA: Morgan Kaufmann.
- Hirsh, H. 1992. Polynomial time learning with version spaces. In *Proceedings of AAAI-92*.
- Hirsh, H. 1994. Generalizing version spaces. *Machine Learning* 17(1):5-46.
- Idestam-Almqvist, P. 1989. Demand networks: An alternative representation of version spaces. SYSLAB Report 75, Department of Computer and Systems Sciences, The Royal Institute of Technology and Stockholm University.
- Johnson, D. S.; Papadimitriou, C. H.; and Yannakakis, M. 1988. On generating all maximal independent sets. *Information Processing Letters* 27(3):119-123.
- Lawler, E. L.; Lenstra, J. K.; and Rinooy Kan, A. H. G. 1980. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*.
- Mishra, N., and Pitt, L. 1997. Transversal of bounded-degree hypergraphs with membership queries. In *Proc. 10th Annu. ACM Workshop on Comput. Learning Theory*, To appear. ACM Press, New York, NY.
- Mitchell, T. 1982. Generalization as search. *Art. Int.* 18:203-226.
- Norton, S. W., and Hirsh, H. 1992. Classifier learning from noisy data as reasoning under uncertainty. In *Proceedings of the National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.
- Norton, S. W. and Hirsh, H. 1993. Learning DNF via probabilistic evidence combination. In *Machine Learning: Proceedings of the Seventh International Conference, 1990*.
- Pitt, L., and Valiant, L. 1988. Computational limitations on learning from examples. *J. ACM* 35:965-984.
- Rivest, R. L. 1987. Learning decision lists. *Machine Learning* 2(3):229-246.
- Smith, B. D., and Rosenbloom, P. S. 1990. Incremental non-backtracking focusing: A polynomially bounded generalization algorithm for version spaces. In *Proceedings of the National Conference on Artificial Intelligence*, 848-853.
- Subramanian, D., and Feigenbaum, J. 1986. Factorization in experiment generation. In *Proceedings of the National Conference on Artificial Intelligence*, 518-522.
- Utgoff, P. E. 1986. *Machine Learning of Inductive Bias*. Boston, MA: Kluwer.
- VanLehn, K., and Ball, W. 1987. A version space approach to learning context-free grammars. *Machine Learning* 2(1):39-74.