Version Spaces, Structural Descriptions, and NP-Completeness

Kevin T. Kelly

Department of Philosophy
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Running Head: Version Spaces and Intractability

Abstract

This paper provides a logical analysis of the problem of version space maintenance which highlights fundamental differences between version spaces of boolean concepts and version spaces for structural descriptions. It then employs the theory of NP-completeness in a heuristic manner to isolate a variety of respects in which the standard version space maintenance procedure is intractable in the case of of structural descriptions. Finally, these results are shown to be relevant to the related task of induction by refinement.

Keywords

- version space
- refinement
- NP-complete
- concept learning
- logic
- search

1. Introduction

In this paper, I examine the problem of version space maintenance (Mitchell, 1982) from the perspectives of formal logic and computational complexity theory. The logical point of view provides a clear setting for concept learning in which to isolate ambiguities in standard statements of the version space maintenance problem. It also highlights fundamental epistemological differences between version spaces over different hypothesis languages.

Computational complexity theory provides powerful techniques for the isolation of distinct sources of intractability that arise in version space maintenance. In this paper, I employ the technique of polynomial reduction in a heuristic manner, to show that the standard version space maintenance procedure can be intractable in at least three distinct ways. First, the test of structural descriptions for consistency with the data is intractable (in two different ways). Second, the problem of deciding entailment among pairs of structural descriptions is intractable (in the same two ways). Third, the fact that many hypotheses can generalize into the same hypothesis gives rise to intractably many repeated tests of the same hypothesis, and the prospects for improving this situation do not appear to be promising. Due to these repetitions, the worst-case performance of the version space maintenance procedure is caiculululated to be vastly worse than that of a simple, non-redundant enumeration of all concepts expressible in the language. These intractability results raise serious questions about the relative efficiency of version spaces as a concept learning strategy.

Finally, the problem of redundant hypothesis considerations is addressed in the related setting of "induction by refinement". In light of this observation, it would make sense for the two research groups to combine forces in addressing the problem.

2. Concept Learning

In the usual concept learning setting, we assume that the learner understands some vocabulary and that he is at some point introduced to a new predicate with which he is unfamiliar. His job is to find a definition of the new predicate in terms of the predicates he already knows so that he will be able to apply the new predicate competently to new cases. Suppose the new predicate is $T(x_v,\dots,x_n)$. Then the learner seeks a *true*^ biconditional formula of the form

$$T(x_1,\dots,x_n) \longleftrightarrow \Phi(x_1,\dots,x_n)$$

where $\Phi(x_1,\dots,x_n)$ is a formula of first-order logic in which only variables $x_1,\dots,x_n$ occur free and in which all vocabulary Items are already understood. For example, the concept "equilateral triangle" can be defined as follows, assuming that the learner understands about triangles, sides of triangles, and their lengths.

Equilateraljriangle(x) <~>
Triangle(x) &
(Ay)(Az)[SkJe_of(y,x) & SkJe_of(z,x) ~> length(y)«length(z)]

Many such biconditional can be true, and the learner is typically considered to be successful so long as he finds any one of them. For example, if the learner understands about angles, he can also define the concept "equilateral triangle" as follows:

EquilateraLtriangle(x) <—>
Triangle(x) &
not (Ey)(Ez)[Ang!e_of(y,x) & Angle_of(z,x) & not measure(z)«measure(y)]

The learner is expected to find a true definition of the target concept on the basis of descriptions of simple facts about particular objects. Since the data is particular and the true definition sought is universal, concept learning is therefore a properly *inductive task.* There are various protocols by which the learner can receive his data. He may be provided with the data, he may ask questions, or he may randomly sample the data.

We speak loosely when we say that the learner's task is to find a definition of *the* target concept. For any particular target concept, no matter how subtle or complicated, there is always *some* learner that can learn it with no effort on the basis of no data whatever. In fact, this learner can be implemented on a cheap electric toaster. Just write the definition on one piece of bread, depress the lever, and in a minute or two a darkened version of the correct answer pops out.

What is not trivial is to learn any one of a wide range of concepts on the basis of evidence about it. No canned, lookup technique like the toaster can solve such a problem. As the learner's range of success becomes wider, the learning problem solved by the learner becomes harder, and the learner seems less "canned" or "toaster-like" accordingly. So a nontrivial concept learning problem is really a *range* of posible target concepts rather than a single target concept, and to solve the problem, the learner must eventually arrive at a definition for any one of these concepts on the basis of evidence concerning it.

We also speak loosely when we talk about "finding" or "eventually arriving at" a true definition since we may have in mind any one of a wide range of precise concepts of convergence, and each of these concepts gives rise to a different learning problem. When a problem is demonstrably unsolvable with respect to a stringent convergence criterion, it is interesting to find slightly weaker criteria under which it can be solved. For example, one might begin with the demand that after some short time the learner output a correct hypothesis and halt. If this is impossible, one might adopt the much weaker criterion that after some time the learner never changes its mind about a correct hypothesis; or one might require, even more weakly, that after some time the learner only produces correct hypotheses; or one might demand only that the learner have a high probability of being nearly correct. The senses of convergence under which a learning problem is solvable or not solvable provide an interesting sense of the intrinsic difficulty of the problem.

In light of these comments, we can logically frame the typical concept learning problem as follows, keeping in mind that convergence can be defined precisely in a variety of ways:

1. **Learner's old vocabulary:** some (finite) set of predicates and function symbols.
2. **Target predicate:** some predicate T not in the learner's old vocabulary. The learner's vocabulary consists of its old vocabulary together with predicate T.
3. **Hypotheses:** formulas of the form

   $T(x_1,...,x_n) \longleftrightarrow \Phi(x_1,...,x_n)$

   such that $\Phi(x_1,...,x_n)$ is a first-order formula whose non-logical vocabulary is included in the learner's old vocabulary and whose free variables are exactly $x_1,...,x_n$.
4. **Learning problem:** a set of relational structures[2] for the learner's vocabulary that agree in domain and in their interpretations of all vocabulary items in the learner's old vocabulary but that differ in their interpretations of T.
5. **Evidence:** Let M be a relational structure in a given learning problem P. A literal is an atomic formula or its negation, and an atomic formula is a predicate followed by the correct number of terms. A set of literals constitutes total evidence for M just in case there is some interpretation of the variables occurring in these literals such that each of these literals comes out true in M and no literal true in M under this interpretation is missing from the set. The evidence for a structure may simply be provided to the learner in increasing finite segments, it may be provided in the form of answers to the learner's specific questions, or it may be randomly sampled. In any event, it is usually assumed that the learner receives only finitely much of the total evidence for the structure at any given stage of inquiry.
6. **The learner's task:** To solve concept learning problem P, the learner must have the following capability: for *each* structure M in P the learner must converge (in some clearly defined sense) to a definition of T that is true in M on the baisis of an evidence presentation for M.

One popular *strategy* aimed at solving concept learning problems is to produce, on any given finite set of evidence, some hypothesis consistent with this evidence. Notice that this strategy is neither a procedure nor the learning problem itself. It is not a procedure, for many distinct procedures making different conjectures on the same data produce only hypotheses consistent with the data. Nor is following this strategy always guaranteed to solve one's learning problem, for some learning problems can be solved without following the strategy and others can be solved by computers only when they do *not* follow the strategy (Osherson, 1986), p. 56.[3]

3. Version Spaces

The version space learning technique is intended as a tactic for producing efficient learners that produce hypotheses consistent with the data. The version space for a set of hypotheses with respect to a given set of data is just the set of all hypotheses in the set whose members are consistent with the data. A version space procedure updates bounds on this set as the evidence increases. Since such a procedure does not produce a single hypothesis, but rather a set of mutually incompatible hypotheses, and since it need not even converge to a unique hypothesis, it is not, itself, a concept learner. It can be converted into a concept learner by clamping on a post-processor that selects some particular hypothesis from the set generated. Another approach is to view a version space procedure as producing the *disjunction* of the definitions in the version space rather than a single definition. The disjunction of the definitions will entail that an object is an instance of the concept just in case each definition in the disjunction classifies it as a positive instance, and will entail that the object is not an instance just in case each definition in the disjunction classifies it as a negative instance. Otherwise, no determination is made.[4] In this circumstance, we can say that the procedure converges to a true definition just in case it reaches a point after which the disjunction is always equivalent to a single true definition (i.e. it classifies all objects correctly).

It is useful to have a precise characterization of how a definition determines a set in a given relational structure. Intuitively, the *defined extension* of the target predicate T in structure M for a given definition of d and a given structure M is just the set of all tuples of objects in the domain of M that are related according to the definition of d. More precisely, the defined extension of target predicate $T(x_1,...,x_n)$ in M with respect to definition d of T is the set of all n-tuples of objects in the domain of M such that when the variables $x_{1f}..jc_n$ are interpreted, respectively, with the objects in the n-tuple, the right-hand-side of the definition d is true in M.

A definition d is said to be *as general* as[5] a definition d' just in case the defined extension of T in M with respect to d includes the defined extention of T in M with respect to d\ In these circumstances, d[f] is as *specific as* d. If the defined extensions are not identical, then d is *more general than* d' and d[1] is *more specific than* d. Definition h is said to be *immediately more general* than definition h* if and only if h is more general than h* and there is no definition h" such that h is more general than h" and h" is more general than h. In this circumstance, h* is sakj to be *immediately more specific ihan* h.

Let V be the version space in a given hypothesis language for some evidence e. An hypothesis h is *maximal* in V just in case no hypothesis in V is more general than h. The subset GEN of V is defined to be the set of all maximal elements of V. An hypothesis h is *minimal* in V just in case no hypothesis in V is more specific than h. The subset SPEC of version space V is defined to be the set of all minimal elements of V. The version space V is determined by GEN and SPEC (i.e. it is the intersection of the downward closure of GEN and the upward closure of SPEC in the "as general as" order). A key idea in version space maintenance is to encode V by encoding GEN and SPEC rather than by listing the much larger set V directly.

As defined, GEN and SPEC are typically infinite because our most familiar examples of hypothesis languages all provide an infinity of logically equivalent definitions which determine the same defined extensions for T in M (and which are therefore all equi-general). Nonetheless, the version space literature speaks of *listing* these sets, which is clearly impossible when they are infinite. What is intended, however, is that the list representing GEN be the result of choosing a unique representative of each *extensions! equivalence class* of hypotheses in GEN, where h and h^f are *extenslonally equivalent* in structure M just in case they define the same extension in M. The same, of course, can be said of SPEC. Unfortunately, there may still be infinitely many extensionally inequivalent hypotheses in GEN or in SPEC[6], in which case the project of listing representatives of equivalence classes in these sets is also doomed to failure.

The standard version space procedure is usually conceived of as receiving a stream of data. Its job is to generate GEN and SPEC in a fixed hypothesis space as new data is presented.[7] The procedure maintains two lists, G and S, of hypotheses that are supposed to contain representatives of all and only extensional equivalence classes of hypotheses in GEN and SPEC, respectively. The procedure proceeds roughly as follows.

1. G is initialized as a list of hypotheses such that for each logical equivalence class of maximally general definitions, some member of this class is in G. S is initialized as a list of maximally specific definitions in the hypothesis language such that for each logical equivalence class of maximally specific definitions, some member of this class is in S. Note that if G or S is infinite, the standard version space procedure cannot even begin.

2. At stage n, we receive new data.

3. If the total data received so far refutes an element h of G,[8] we apply a *specialization*

operator to h to generate a set of hypotheses weaker than n, and then test these, specializing any new failures until we arrive at only unrefuted hypotheses.

4. If the data refutes an element of S, we successively apply a *generalization operator* until we reach unrefuted hypotheses. In either case, the refuted hypotheses in G are replaced with their unrefuted, specialized versions, and the refuted hypotheses in S are replaced with their unrefuted, specialized versions.
5. Next, we compare the sentences in G to weed out redundancies. If one hypothesis in G is found to be more general than another hypothesis in G, then the less general one is eliminated from G. If a pair is found to be extensionally equivalent, then one of them is deleted.
6. Similarly, if one hypothesis in S is found to be more specific than another, then the less specific one is eliminated from S. If a pair is found to be extensionally equivalent, then one of them is deleted.
7. The resulting sets G and S are output, and new data is called for.

Specialization and generalization are not defined independently of particular hypothesis languages in the version space literature, but the following characterization seems to fit. Let h be a definition. Then gen(h) denotes some set X of definitions immediately more general than h such that for each logical equivalence class of definitions immediately more general than h, some representative of this class is in X. We can characterize spec(h) similarly, by substituting "immediately more specific" for "immediately more general".

When only finitely many extensions are definable for T in M, the version space procedure will correctly maintain G and S so that G=GEN and S*SPEC, so long as the required consistency and entailment tests and the operators gen and spec are computable. Moreover, the version space bounded by G and S will eventually contain only correct definitions in the limit (although the procedure will not necessarily be able to know when it has succeeded).[9] But when infinitely many extensions are definable for T in M, the version space algorithm can run into difficulties. For example, it is possible in such a space for gen(h) to be empty when no definition is *immediately* more general than h. For example, consider the space of all hypotheses of the form "T(x) <~> x has at most n parts and x is red" together with the hypothesis T(x) <—> x is red". The latter hypothesis has no immediate generalization in the overall hypothesis space since there is no finite upper bound on the number of parts a thing can have. It is also possible for gen(h) to be infinite. For consider the space of hypotheses of the form "x has n parts" where n denotes some natural number, together with the single, maximally general hypothesis "x«x". Every other hypothesis in the language is immediately more specific than "x=x" in the hypothesis space, and if objects of every size exist in M, then each of these immediately more specific definitions defines a distinct extension for T in M. When gen(h) is empty but there exists an hypothesis consistent with the data, then the set G no longer agrees with GEN, and the version space procedure fails. When gen(h) is infinite, the version space procedure stalls forever in building G, and fans to make an output. If it chooses a finite subset, then it may fail to maintain agreement between G and GEN. In either case, the procedure may fail to converge to a set of correct definitions.

Recall that the relation "as general as" is defined in terms of inclusion between extensions definable in the underlying structure. It is important to understand the relationship between this relation and the relation of logical entailment between right-hand-sides of definitions. Recall our two definitions of the concept "equilateral triangle". These definitions are not *logically equivalent,* but they are coextensive in the world of plane figures, sides, angles, and their measures. In general, if the right-hand-side of definition d entails the right-hand-side of definition d', then d[f] is as general as d. But the converse is not true, d* can be as general as d even when the right-hand-side of d does not entail the right-hand-side of d\ as in the case with the equilateral triangle example. Hence, unless one considers entailments in a sufficiently strong, true, substantive theory, searching the structure of entailment over logical equivalence classes of definitions need not be the same thing as searching the structure of inclusion over the extensions definable for T.[10] On the other hand, GEN and SPEC will still be maintained properly even if the entailment order is searched instead of the order of inclusion over defined extensions. The former search space fails to have certain arcs that the latter search space has, so the trace of the search is different, but there remains some search path to each hypothesis we would have reached in the more richly connected space.

4. Version Space Maintenance and NP Completeness

Since the general version space method is recommended on the basis of its increased efficiency in certain applications, the question of its computational complexity in various hypothesis spaces is crucial. An indispensable tool in this sort of inquiry is the theory of NP completeness. Haussler (Haussler, 1987), and Pitt and Valiant (Pitt, 1986) have shown that for various, commonly examined hypothesis languages, the problem of determining whether an hypothesis in that language exists that is consistent with given data is NP-complete. These results show that the problem of determining whether the given sets G,S are, respectively, the sets GEN and SPEC for given set of data is NP-complete for the hypothesis languages considered. Hence, if P is not equal to NP, then the time spent in generating GEN and SPEC from data does not grow polynomial^ in the size of the data, and in the sizes of the sets GEN and SPEC to be generated.[11] Although a single NP-completeness proof has fundamental theoretical significance for

a computatfonal problem, there is practical, heuristic value in locating alternate sources of intractability in the problem. Different reduction arguments can isolate very different aspects of intractability, and some of these may have more practical significance in the short run than others. In what follows, I examine such additional sources of intractability for the case of maintaining **version spaces of** structural descriptions.

5. Version Spaces for Structural Descriptions

A *structural description* is just **a** definition of the form

$T(x) <\sim> Qx_l...Qx_n<J>(x,x_1,..._lx_n)$

where the Q's are quantifiers, x is **a** finite vector of variables, and <!Hs **a** quantifier-free formula, possibly with relational predicates and function symbols. Purely existential structural descriptions are structural descriptions in which only existential quantifiers occur. Such concepts are familiar in the machine leamiog literature (Winston, 1975) (Haussler, 1987). A typical example of an existential structural description is

Arch(x) <~>
(Ew)(Ey)(Ez)
[Part_of(w,x) &
Part_of(y,x) &
Part_of(z,x) &
«Touch(w,y) &
Onjop(z.w) &
Onjop(z.y)]

A universal structural description is **a** structural description whose quantifier prefix involves only universal quantifiers. Universal structural descriptions are interesting in their own right, and are also the complements of the commonly studied existential structural descriptions of the sort popularized by Winston (Winston, 1975).

Typical examples of such concepts include:

Blocks World

Foundation-of(x,y) <--> (Az)[(Part-of(z,y) & not z=x) —> Below(x,z)]

Arithmetic

Prime(y) <—> $(Ax_1)(Ax_2)[-x_lx_2=y \vee x,«y \vee x_2=y]$

Minimal(y) <—> $(Ax)[x\geq y]$

Much of the discussion of version spaces in machine learning has centered on *monadic boolean* concepts, or quantifier-free concepts involving only unary predicates and no function symbols (Mitchell, 1982). Version spaces for structural descriptions differ in three important respects from those for monadic boolean concepts.

The first difference is that there are infinitely many distinct logical equivalence classes of concepts with existential quantifiers, and the same can be said when the concepts have universal quantifiers. In contrast, the set of all logically distinct concepts with boolean connectives and finitely many unary predicates is finite.

The second difference is that all the facts concerning an individual in a monadic language can be characterized by a finite set of literals such that for each predicate in the language, it is either asserted or denied of the individual's name by some literal in the set. Such a set is called an *instance* or a *state description*. With relational predicates, no finite set of literals can summarize an individual's relations with all other individuals in an infinite universe. So there is no notion of finite positive and negative "instances" when relational predicates are in the hypothesis language.

Winston's "semantic networks" are straightforward recodings of first-order relational formulas and Winston speaks of positive and negative instances as though they are finite. This does not contradict what I have just said, because Winston assumes special restrictions on the the relational structure under study, on the range of concepts to be learned, and on the data presentation. Suppose there is some relation R denoted by predicate P such that for each object d at most finitely many objects d' bear R to d. Suppose also that it is known *a priori* that the target predicate T(x) can be defined in terms of relations among objects bearing R to x. Finally, suppose that the data presentation provides a flag "done for x" signalling when variables standing for all objects bearing R to x occur in the data. When all these assumptions are met, we have all the data we need about x when we know how the objects we have seen so far relate to it and to each other. In particular, once we see the flag we know the *universal fact* that no object we have not already seen bears R to x. Hence, this setting obviates the need for negated atoms concerning relation R, which is why the data to Winston's system does not explicitly encode the fact that x is *not* a part of y.

But when the relations of an object with infinitely many others are relevant to its definition (e.g. consider the concept "least-number") or if we cannot be sure that *all* the finitely many relevant objects are mentioned at once (e.g. consider the case of definining species by anatomical features which may go unnoticed at first) then we must still save the old data. Hence, the kind of incremental version space procedure that relies just on the current sets GEN and SPEC and the current, finite state description cannot be expected in the general case of structural descriptions. The old data has to be saved, for it contains the initial chunks of infinite descriptions of positive and negative examples of the target concept.

The third difference between version spaces of structural descriptions and version spaces of boolean concepts concerns the togical relevance of positive and negative examples of the target predicate to a proposed definition of this predicate. In monadic boolean version spaces,[12] the set GEN is driven downward by negative examples and the set SPEC is driven upward by positive examples. In contrast, only negative examples can be inconsistent with an existentially quantified concept, and only positive examples can be inconsistent with a universally quantified concept. To see this, note that particular data can refute but not verify a universally quantified sentence and can verify but not refute an existentially quantified sentence. To refute a definition, one must find data that refutes one side and verifies the other. If the definition is existentially quantified, (i.e. of form C(x)<—>Ey<J>(x,y)) we can possibly find objects b,c such that -C(b) and *(b,c) are true, and which falsify the definition. In this case, b is a *negative example* of C, since C does not apply to it. But notice that a positive example could never be inconsistent with C(x)<—>Ey<X>(x,y), for no number of objects failing to satisfy <t>(x,y) could guarantee that objects satisfying 4>{x,y) do not exist. Hence, an existential concept can be refuted only insofar as its extension is too big, so only the set GEN must be updated as data comes in. SPEC is never altered.

Similarly, if the definition is universally quantified (i.e. of form C(x)<~>Ay<t>(x,y)) then a pair of objects b,c such that C(b) and -4>(b,c) are true falsify it, where c is a *positive example* of C. Again, a negative example could never refute such a definition for no finite data can ensure that the universal right-hand-side is true. Finally, these observations tell us that a universal concept can only be refuted insofar as its extension is too narrow, and so only the set SPEC is altered as data comes in. GEN is never altered.

Finally, neither positive nor negative examples can refute a definition essentially involving mixed quantifiers, for no particular data is inconsistent with such a formula. So in a space consisting only of such hypotheses, neither GEN nor SPEC would ever be updated on any data. For example, the concept

is-an-immediate-predecessor(x)<—>(Ez)(Ay)[z>x & (-y>x v -z>y)]

is not inconsistent with any finite set of data. In such cases, there seems little point to version space maintenance.

Notice that the preceding observations concerning consistency with the data may fail in Winston's setting which provides flags to signal that all objects related to a given object have been seen. In this case, the data essentially communicates universal truths to the learner which may be inconsistent with definitions involving mixed quantifiers. In such settings, both GEN and SPEC will be altered in the course of inquiry.

6. Testing and Comparing Hypotheses

Recall that the usual version space maintenance algorithm decides two logical relations. First, it *tests* hypotheses for consistency with the evidence, and second, it decides entailment between pairs of hypotheses to weed non-minimal hypotheses out of GEN and non-maximal elements out of SPEC. It is therefore natural to ask how difficult it is to decide these logical relations for in the case of universal structural descriptions.

It is easy to see that the consistency problem is at least NP-hard in general, since we can take the data set to include just one atom T(a), where T is the target predicate, and encode each instance of propositions! DNF satisfiability as a definition of T whose right-hand-side is a closed universal formula with only vacuous quantifiers. If the right-hand-side is a satisfiable boolean formula then the definition is consistent with the data. Otherwise, it is not, for it excludes a positive example of T. Since DNF satisfiability is intractable, the problem of testing structural descriptions is as well. This observation shows that intractability can arise simply in light of the boolean connective structure of the hypothesis under test. But there is no reason to stop here. There may be intractable features of the problem even if we restrict the connective structure in such a way as to knock out the previous result. Or to put it another way, the the consistency problem may be intractable in very different ways, and we doni properly understand the problem from a computational perspective until we have investigated a variety of reductions depending on different aspects of the problem.

In the case of testing structural descriptions against the data, there is indeed an alternate source of intractability that can remain even when we restrict the problem to undercut the reduction of propositions! satisfiability (e.g. by eliminating negation). Intuitively, the difficulty is that there are intractably many ways to substitute variables in the data for variables in the hypothesis under test when seeking a counterinstance of the right-hand-side of a universal definition.[13] And, under the usual hypothesis that P is not NP, there is no quick way to sift through them.

Checking entailment among right-hand-sides of concept definitions to determine whether one concept is more general than the other is also probably intractable. I will show the result for entailment first, and the inconsistency result will drop out as a corollary.

Formally, let a prenex normal formula be called *simple* if and only if it is a universally quantified disjunction of atoms free at most in variable x and involving only ternary predicate P. For example:

(Ay)(Az)(P(z,z,x) v P(y,x,x))

is simple, while

(Ay)(Ez)(P(x,x,z) v P(y,z,x))

(Az)(-P(x,z,z))

(Ay)(P(y,z))

are not. (Note that since simple formulas do not permit negation, the simple reduction of DNF satisfiability just discussed will no longer work).

Then we have the following result (Kelly, 1986):

Theorem 1:[14]

Let S be the set of all simple formulas. Let c be the formula (Ay)(Az)(Aw)(P(x,y,z) v P(x,z,w) v P(x,w,y)). Then deciding whether an an arbitrarily chosen h in S entails c is an NP-complete problem.

Corollary 1:

Let e be the data set {T(a),-P(a,b,c),-P(a,c,d),-P(a,d,b)}. Then deciding whether an arbitrary h in S is consistent with e is an NP-complete problem.

Proof of theorem: First we need a definition and a lemma:

Let C, C' be function-free clauses.[15] Then C *collapses into* C' if and only if there is a substitution θ of variables for variables such that each literal L occurring in Cθ occurs in C'.

Lemma a: If h, h' are simple formulas, then h |= h' if and only if h collapses into h'. A proof of the lemma is provided in (Kelly, 1986).

It is therefore easy to see that the simple universal formula entailment problem is in NP, since one can guess a substitution θ and verify it in polynomial time. So it remains only to show that the problem reduces an NP-hard problem in polynomial time.

The problem of deciding whether an arbitrary, non-directed graph is 3-colorable is NP-complete (Stockmeyer, 1973). A non-directed graph is a pair <V,R> where R is a set of unordered pairs drawn from V. G is 3-colorable if and only if there is an f:V-->{1,2,3} such that for any x,y in V, f(x)=f(y) only if {x,y}εR. Let G,G" be two graphs <V,R>, <V',R'>, respectively. Say that G *collapses into* G' if and only if there is a function f from V to V' such that for each x,y,εV, if {x,y}εR then {f(x),f(y)}εR'. Finally, let K3 be the graph

$\langle V^*, R^* \rangle$, where $V^*=\{1,2,3\}$, and $R^*=\{\{1,2\},\{2,3\},\{1,3\}\}$.

Lemma b: G is 3-colorable if and only if G collapses into K3.

===> Suppose $G=\langle V,R \rangle$ is 3-colorable. Then there is an $f:V \to V^*$ such that for each $x,y \in V$, if $\{x,y\} \in V$ then $f(x)$ is not identical to $f(y)$. For reductio, suppose that there are $x,y \in V$ such that $\{x,y\} \in R$ but $\{f(x),f(y)\}$ is not an element of R^*. Since the range of f is a subset of V^*, and for each a,b in V^* such that a is not identical to b, $\{a,b\} \in R^*$, it follows that $f(x)=f(y)$. So f is not a 3-coloring, which is a contradiction.

<=== Suppose G collapses into K3. Then there is an f such that for all $x,y \in V$, if $\{x,y\} \in R$ then $\{f(x),f(y)\} \in R^*$. For reductio, assume that $\{x,y\} \in R$ but $f(x)=f(y)$. But since no pair $\{x,x\}$ is in R^* for any $x \in V^*$, it follows that $\{f(x),f(y)\}$ is not in R^*, which is a contradiction. This concludes the proof of the lemma.

Now it suffices to show that for any non-directed graph G, we can produce in polynomial time a pair of simple formulas C,K, such that C entails K if and only if G collapses into K3. Our strategy is to encode vertices as quantified variables and literals as unordered pairs of these variables. We encode K3 as the simple formula

CK3: (Ay)(Az)(Aw)[P(x,y,z) v P(x,z,w) v P(x,w,y)].

Next, we construct a simple formula C[G] corresponding to arbitrary graph $G = \langle V,R \rangle$ as follows. Enumerate V. For each edge {y,z} in G, put the disjunct P(x,y,z) into C if y is prior to z in the enumeration, and add the disjunct P(x,z,y) otherwise. Clearly, this can be done in time polynomial in the size of G.

Lemma c: C[G] collapses into CK3 if and only if G collapses into K3.

<=== Suppose G collapses into K3. Without loss of generality assume that V, V^* are disjoint (graph 3-colorability is still NP-complete if V and V^* are always disjoint). Then there is an $f:V \to V^*$ such that for all $y,z \in V$, if $\{y,z\} \in R$ then $\{f(y),f(z)\} \in R^*$. Let g be the map from vertices to variables defined in the construction of C[G] and CK3. Note that g is 1-1 since each vertex is encoded by a distinct variable. Define substitution θ so that $x\theta=x$ and if y is not identical to x, then $y\theta=g(f(g^{-1}(y)))$. Suppose for contradiction that θ does not collapse C[G] into CK3. Then there is a literal in C[G] such that $C[G]\theta$ does not occur in CK3. Since all literals in C[G] and CK3 have x in the first position and only in first position, and since $x\theta=x$, there exist variables y,z such that P(x,y,z) occurs in C[G] and $P(x,y\theta,z\theta)$ does not occur in CK3. Hence, $\{g^{-1}(y),g^{-1}(z)\} \in R$ but $\{f(g^{-1}(y)),f(g^{-1}(z))\}$ is not in R^*, which contradicts the fact that f collapses G into K3.

===> Suppose C[G] collapses into CK3. So there is a substitution g(q) such that for each literal L occuring in C[G], $L\theta$ occurs in CK3. Let g be defined as before. For any vertex d in V,V^*, define $f(d)=g^{-1}(g(d)\theta)$. Suppose for contradiction that f does not collapse G into K3. Then there are vertices d,d' in V such that $\{d,d'\} \in R$ but $\{f(d),f(d')\}$ is not in R^*. So P(x,g(d),g(d')) occurs in C[G] but $P(x,g(d)\theta,g(d')\theta)$ does not occur in CK3, which contradicts the fact that θ collapses C[G] into CK3.

So we have C[G] entails CK3 if and only if C[G] collapses into CK3 (lemma a); C[G] collapses into CK3 if and only if G collapses into K3 (lemma c); and G collapses into K3 if and only if G is 3-colorable (lemma a). Q.E.D.

Proof of Corollary:

Instead of generating C[G] and CK3 for graph G, we generate e = {T(x),-P(x,y,z),-P(x,z,w),-P(x,w,y)} together with the definition D = 'T(x) <---> C[G]'. This reduction is clearly polynomial time computable. e descibes a positive example of the target concept T(x), so if C[G] is inconsistent with the existential closure of the conjunction of the remaining literals, definition D is refuted, for it misclassifies a positive example. But C[G] is inconsistent with E&e if and only if C[G] entails -E&e, which is equivalent to CK3. So e refutes C[G] if and only if C[G] entails CK3. Q.E.D.

Theorem 1 and its corollary show that the standard version space procedure encounters intractability when it appeals to subroutines that decide consistency with the data or that compare two hypotheses for entailment in order to weed out redundancies, even over hypotheses with a very simple connective structure. The intractability indicated in these theorems will not show up in toy cases. For example, if there are just five bound variables in the concept under test, the possible instantiations can be readily exhausted on a computer (there are but 52 of them altogether). But in real applications (e.g. the carpenter's structural description of a kind of desk having a top, two sides, four legs, and five drawers-each with three sides, a front and a bottom) we can easily exceed thirteen bound variables. In this case we really feel the pinch, for there are already 190,899,322 possible instantiations. (A more detailed combinatorial analysis of the number of substitutions will be presented in the next section).

7. The Problem of Repeated Hypothesis Examinations

As we have seen, the standard version space procedure applies a generalization operator to a failed hypothesis, to the generated daughters of this hypothesis that fail, and so forth, until no more failures are encountered through generalization. If we think of paths in this tree of generalizations as a search tree, we can envision the search as splitting apart the inclusion order of extensions definable in the hypothesis language into a search tree. The fact that many distinct hypotheses can generalize into the same hypothesis gives rise to two seemingly unattractive features in the standard version space maintenance procedure. First, several refuted concepts may -generalize" to a given, wider concept, and this wider concept would be examined, tested, added to SPEC and deleted from SPEC one time for each of its refuted parents. Second, a refuted concept may "generalize" to a concept generalizing an unrefuted concept. Such a concept will be added to and then deleted from SPEC to no apparent advantage. And we have just seen that the entailment check involved in the elimination of these redundancies can be very costly. Both observations hold as well for specialization. Even if a comprehensive list of tests is saved so that repeated tests are not performed, the standard version space maintenance method still faces repeated considerations of the same concept and repeated visits to the list of prior tests. Such consultations still soak up computational resources.

These considerations raise a natural, practical question: is there an easy technique that exploits the logical structure of the hypothesis space to avoid these redundant hypothesis considerations in the case of structural descriptions? To examine this issue more carefully, let us consider a tidy, finite subspace of the infinite set of all universal concept descriptions on a given vocabulary. It turns out that each observation we make concerning this space holds equally well for the corresponding set of existentially quantified concepts if we exchange NEG for POS, GEN for SPEC, and logical strengthening for logical weakening. (Recall that for universal concepts POS forces the weakening of SPEC and for existential concepts NEG forces the strengthening of GEN).

Consider the space of all concepts that result from substituting quantified variables for quantified variables In a given concept definition. In particular suppose our system can understand an n-ary predicate P. Then we are interested In In the class C(n) of all the concepts that can be expressed by filling in the #'s with universally quantified variables in the following concept template.

T(y) <--> (A*)...(A-)[P(y.\V.... #)]

Notice that the only equivalent concepts in C(n) are those that result from one another by substituting bound variables for bound variables. For example, the concepts

$T(x) <\sim> (Ax_1)(Ax_2)(P(y,x_1,x_2))$

$T(x) <\sim> (Ax_l)(Ax_2)(P(y,x_2,x_1))$

are logically equivalent and result from the permutation of x_1 with x_2.

The lattice of all logical equivalence classes of concepts in C(n) ordered by entailment among their right-hand-sides is isomorphic, or structurally identical, to the lattice Part(n) of refinement over the set of all partitions of a given set of cardinality n, as is illustrated in the figure (Aigner, 1979). To see the isomorphism, think of distinct variables as naming partition cells, and of variable positions in the concept (e.g. first occurring variable, second occurring variable, etc.) as corresponding to objects in the partitioned set. Then the order of variable occurrences in the formula determines a set partition of the variable positions, and a partition of these positions determines a variable pattern. This is a useful observation, because the combinatorial properties of partition lattices are well known.

The first thing we know is that the number of logical equivalence classes in C is B(n), where n is the number of asterisks to specify and B(n) is known as the nth Bell number (Aigner, 1979). From the figure, it is evident that a level in the lattice corresponds to the number of distinct bound variables occurring in the concept. The number of equivalence classes of concepts at the kth level of the lattice C(n) is S(n,k) where S(n,k) is the Stirling number of the second kind. One recurrence relation for S(n,k) is the following:

$S(n,n)=1$

$S(n,0)=0$

S(n,k) « S(n-1,k-1) + k(Sn-i,k)

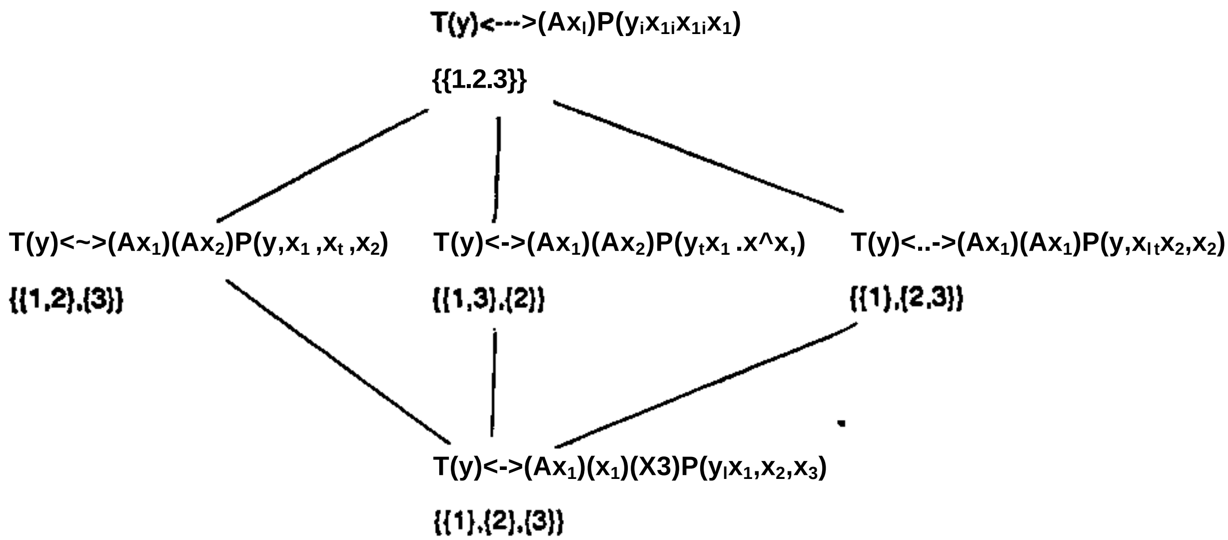

Figure 1: partitions and structural descriptions

It is easy to see that for each $k \leq n$, $S(n,k) > k^{n*k}$ (just neglect the left-hand addend in the inductive clause of the recurrence and count the number of factors k one has by the time the basis condition for S(n,n) is met). Hence, S(n,k) is exponential in n for any fixed k.

We have seen that the standard version space procedure's repeated application of the specialization operator to failed hypotheses splits the lattice of equivalence classes in C(n) from bottom to top into a search tree.

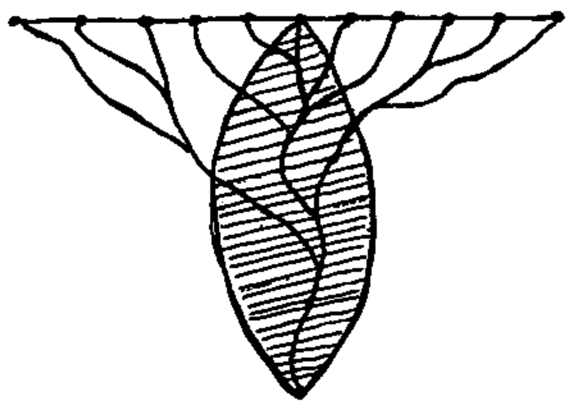

Figure 2: lattices
and search trees

Such a search encounters the same hypothesis once for each distinct path from the lattice bottom to this hypothesis (in the worst case in which the top of the lattice is refuted). It is therefore interesting to consider the number of paths from the bottom of the lattice to a particular node.

It is well known (Aigner, 1979) that the downward closure of an element *K* in a partition lattice is the lattice product

$$\Pi_{c \in x} \mathrm{Part}(|c|).$$

That is, the downward closure of an element in a partition lattice is a lattice product of partition lattices. To see this, observe that each cell of a partition can be split up independently of any other. Hence, each cell can be viewed as the top of its own partition lattice, and since the refinements of each cell are independent from those of any other cell, the set of all refinements of the original partition is the lattice product of the refinement lattices for each cell of the original partition.

Hence, the number of paths from the lattice bottom to a given lattice element is the set of all paths from the bottom of a product of partition lattices to its top. This quantity is messy to express for arbitrary partition lattice products.[16] But for concepts of the special form

c(k):

$$T(x) \leftrightarrow (Ax_l)\ldots(Ax_k)[P(y_t x_{li} x_l, x_1, x_{1l} \ldots {}_f x_2, x_3, \ldots x_k)]$$

the downward closure of c(k) in Part(n) is isomorphic to the partition lattice Part(n-k). Hence, the number of distinct paths to c(k) is just the number of paths from the top to the bottom of Part(n-k). Let (n,k) be the binomial coefficient. Then the number of paths from the bottom to the top of Part(n) is just[17]

$$\#\mathrm{paths}(n) = (2,2)(3,2)(4,2)\ldots(n,2) = [n!(n-1)!]/2^{n-1}.$$

#paths(n) is exponential, for it is almost everywhere bounded below[18] by 2^{2n*3}.

The number of paths to c(k) from the bottom of Part(n) is

$$\#\mathrm{paths}(n,k) = \#\mathrm{paths}(n-k)$$

which is also exponential in n for fixed k, since #paths(n-k) is.

The following table compares the number of hypotheses in Part(n) with the number of distinct paths through the lattice the version space algorithm pursues in the worst case in which the top of the lattice is refuted. When the initial evidence POS refutes the weakest hypothesis in the lattice, the time required to traverse all these distinct paths to the lattice top is worse than the time required to sort the entire hypothesis space into refuted and non-refuted hypotheses by simple enumeration. Once this fact

n	# of hypotheses in C(n)	# of paths in Part(n)
1	1	1
2	3	1
3	5	3
4	15	18
5	52	180
6	203	2,700
7	877	56.700
8	4,140	1,587,600
9	21,147	57,153.600
10	115,975	2,571,912.000
11	678,570	141,455,160.000
12	27,644,437	9.336.040.560.000
13	190,899.322	728.211,163,680.000

Figure 3: number of hypotheses vs. number of search paths

becomes apparent, it is simple to alter the standard version space algorithm to sidestep it. If we clean out redundancies *while searching for* unrefuted hypotheses, then we can search the lattice without traversing all possible search paths to the hypotheses sought and without any risk of missing these hypotheses, at the smaller expense of searching for and eliminating redundancies in SPEC as we go.

But there still remains the difficulty that many hypotheses may generalize to the same hypothesis, even when we choose a unique representative of each logical equivalence class. In particular, consider the number S(n,2) of hypotheses that have the lattice top as an immediate generalization in Part(n). Since $S(n,k)>k^{n,k}$, S(n,2) is exponential in n.

n	S(n,2)
1	1
2	1
3	3
4	7
5	15
6	31
7	63
8	127
9	255
10	511
11	1.023
12	2,047

Figure 4: number of immediate descendents of the top of Part(n)

Since our worst-case estimate is based on the number of hypotheses that generalize to the lattice top, one might propose a special heuristic in the version space maintenance procedure that stops generalization when the search has reached the second level of the lattice. But this doesn't solve the problem in general, for the number of partitions generalizing into (i.e. that are refinements of) a given partition $\pi\varepsilon C$ is exponential in the cardinality of the largest cell in π, by similar reasoning. Since quantified structural descriptions can be viewed as finite set partitions, the worst-case number of hypotheses that can generalize into the same hypothesis rises exponentially in the maximum number of bound variables characterizing the space of quantified structural descriptions.

8. Prospects for Improvement

The standard version space maintenance procedure proceeds locally, and never stops to consider whether the generalization of one failed hypothesis is already covered by another hypothesis in SPEC, or whether two failed hypothesis generalizations generalize to the same hypothesis. Hence, it can consider the same hypothesis in the lattice Part(n) exponentially often in n. The question then arises whether the mathematical structure of partition lattices could be exploited by a more efficient algorithm that fails to consider the same hypothesis intractably often in n. One natural approach would be to try to develop a relatively elegant procedure that solves the following generation problem:

Given:
n
$k \leq n$
disjoint subsets SPEC, FAIL of Part(n)

To generate:

the set of all partitions
at level k of Part(n) that are

(a) in the upward closure of FAIL
(b) not in the upward closure of SPEC

if there is such a partition, and the answer "no" otherwise.

The cardinality of the set to be generated in solving this problem can already be exponential in the sizes of the inputs. For example, when FAIL and SPEC are empty, then we generate all of level k of Part(n), whose cardinality is $S(n,k) > k^{n-k}$. Hence, it is trivial that no algorithm can solve the problem in polynomial time.

But there is no reason to stop at this trivial observation. The machine learning community has already acclimated itself to the idea of exponential version space maintenance procedures. The question is, rather, one of making an admittedly exponential procedure more elegant. One candidate for what is meant by "elegance* here is that no more than polynomial time be spent *per hypotheis ouput* by the generation algorithm. Even if the set to be generated is exponentially large, it is all the worse if the average time spent per hypothesis in the set generated rises exponentially in the worst case.[19]

Let GEN be an algorithm that solves the problem in question. If GEN is itself elegant, then we can maintain SPEC on new data POS in the following, elegant way:

```
UPDATE (n, SPEC, POS)
     Initialize FAIL, EXAMINE to the empty set;
     Add POS to the old data;
     Move all refuted elements of SPEC from SPEC to FAIL;
     Let k be one greater than the lowest level
          element of FAIL;
     Until GEN(n, k, SPEC, FAIL) is empty do
     Begin
          Set EXAMINE :* GEN(n, k, SPEC, FAIL);
               (EXAMINE now contains all hypotheses
               at level k of Part(n)
               that are not enailed by any
               element of SPEC but that are entailed
               by the refuted hypotheses now in FAIL);
          Test each member of EXAMINE, placing the ones
               that pass in SPEC and eliminating
               non-minimal hypotheses in the result;
          Set FAIL, EXAMINE to the empty set;
          Set k:« k+1
     End.
```

The operation of UPDATE is illustrated in the accompanying figure.

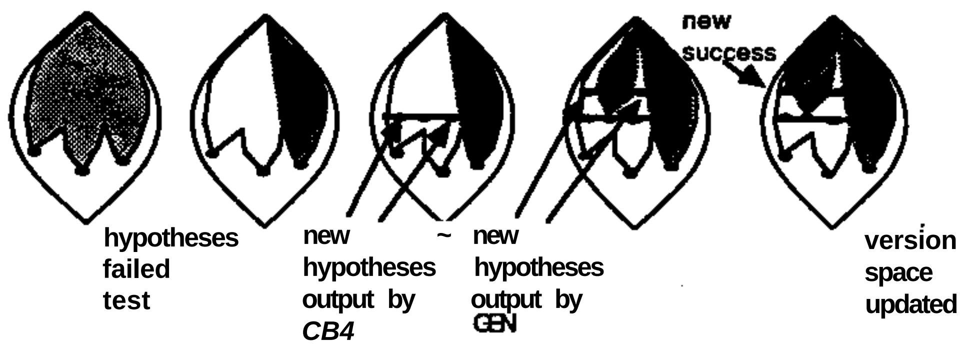

Figure 5: the operation of UPDATE

When an element of SPEC fails, UPDATE tests only those hypotheses in Part(n) that are no longer covered by any element of SPEC and that have not been tested and found to fail earlier.

But unfortunately, there is probably no implementation of the subroutine GEN that does not somehow look at many more hypotheses than are actually output. Or to put it another way, some hypothesis in the output takes more than polynomial time to generate, relative to the sizes of the inputs SPEC, n and k. For suppose that GEN spends no more than polynomial^ much time finding each answer it generates, including the "no" answer. Then GEN* can be used as a polynomial time solution to the following problem:

The Partition Lattice Upward Closure Complement Search (PLUCCS Problem)

Given:
n
k
a subset S of Part(n)

To decide:
Whether there is a partition no lower than level k that is not in the upward closure of S in Part(n).

Since GEN spends no more than poiynomially much time per hypothesis generated, there is a polynomial p such that for every instance labelled "no", the "no" is found in time p from the inputs. By simulating GEN(n,k,S,{}) and p on the size of <n,k,S,{}>, we can answer "no" if GEN says "no" under bound p, and we can stop the computation and answer "yes" otherwise. This overall procedure would be a polynomial time solution to the PLUCCS problem.

But if P is not equal to NP then there exists no such solution because

Theorem: The PLUCCS problem is NP-complete (Kelly, 1986).

Proof:

First PLUCCS is in NP, for consider a nondeterministic machine that checks whether S is empty. If S is empty, the machine immediately answers yes. If S is non-empty, then the machine non-deterministicaliy guesses a partition *n* and verifies in time polynomial in the size of <S,n,k> that *n* is at level k of Part(n) and that *n* is not a refinement of any element of S.

The non-trivial part is to show that UCS is NP-hard. This is accomplished by showing that *The Minimal Set Cover (MSC) Problem* reduces to the UCS problem in polynomial time.

The Minimal Set Cover (MSC) Problem

Given:
a finite set R;
a subset *T* of the power set of R;
a natural number c.

To Decide:
there is a subset *V* of *T* such that *\r.≤c* and Ur-R

The MSC problem is known to be NP-Complete (Garey, 1979).

Let <R.r.o be an arbitrary instance of MSC. Let r:-|R| and g>|G|. Assume an arbitrary enumeration {G,....G^Jofr. Now define:

- •t(I\g.x) > the refinement n of {{1.2}.{3,4}....{2g-1,2g}} such that for each i from 1 to g, 2i is separated from 2i-1 in x just in case $xcG_{(}$.
- AXK)MS(g):-the set of all partitions *K* of {1....2g} such that exact* o n # ^{IB of distinct numbers from 1 to 2g are in the same celt of ic and this unique pair also satisfies the property that there is no k such that {i,j}-{2k,2k-1}.

We now define our proposed reduction of MSC to PLUCGS as follows.

f(<R,r.o)*<n_vk,S> where

n:« 2g

k:*c+g

S:« AXIOMS(g) U {t(T.x): xcR}

Note that AXIOMS(g) can be generated in time polynomial in g-IH because each partition in AXIOMS(g) can be generated by a simple choice of a pair 1≤i<j≤2g such that there is no k such that j«2k and i«2k-1. But there are no more than 2g(2g-1)/2 pairs altogether, and those for which there is such a k can be quickly weeded out ft is easy to verify that the rest of the operations are polynomial in the size of <R,r,o. Hence, f can be computed in time polynomial in the size of <R,F.c>.

h remains to show thai <R,r,c>cMSC if and only if f(<R,r_tc)«(n,k,S)cPLUCCS.

«.«> Suppose <R,r,OEMSC. Hence, there is a subset C of *T* such that |C|≤c and UC-R. Define $r._c$ to be the refinement of {{1,2},{3,4}$_f$...{2g-1,2g}} such that for each i from 1 to g, 2i is separated from 2i-1 in n_c if and only if GJEC. Since |C|<c, n_c is at level no lower than g+c since $7t_c$ is defined as a refinement of a partition at level g that has at most c more cells split No element of AXIOMS(g) refines $r._Q$ because each axiom 7i has a eel! {t.j} that is split in *n^* because * refines {{1,2},{2,3}...,{2g-1,2g}} by definition. Finally, let xeR. Since UC«R, it follows that there is a G_reC such that *$XEG_{(}$.* Hence there is an i between 1 and g such that Ji, separates 2i-1 from i while t(T,g_fx) puts 2i-1 in the same cell as 2i. So t(T,g,x) is not a refinement of n_c. Since x is an arbitrary element of R, n_c is refined by no element of {t(T,g,x):xER}.

<««« Suppose f(<R_fr,o) « <n,k,S> - <2g_fg+c,AXIOMS(g)U{t(T_fg,x)DceR}> £ PLUCCS. Hence, there is a partition *n* at level k«g+c or above such that *n* is refined by no element of AXIOMS(g) and by no t(F,g_tx), for xeR. Since *n* is refined by no element of AXIOMS(g), *n* must be a refinement of {{1,2}$_f${3,4}$_f$...$_f${2g-1,2g}}. Define C^G^i-i,2i}oc}. Let xeR. Since *n* is not refined by t(r,g_fx), there is an i such that {2i-1_t2i}Et(r,g,x) and 2i-1_f2i are separated in *n.* Hence, *xzGfiC^.* Since x is an arbitrary element of R, UC«R. Finally, since JI is at or above level k«g+c. and *K* is a refinement of {{1,2},{3,4}....{2g-1,2g}}, and since the remaining c cuts are the only ones involved in adding G's to C^ |C|≤c. Q.E.D.

A suggestive, if not rigorous, reading of this result is that somehow the intractable branching performed by the standard version space maintenance algorithm will be simulated by GEN in any allegedly improved procedure organized like UPDATE. Hence, the result suggests that there is little to be gained at present by applying standard artificial intelligence techniques to improved version space maintenance algorithms of this sort.

To understand the limitations of the above result, let us consider the problem of boolean version space updating. Boolean version spaces are easy to update without a branching search. Initialize SPEC as a contradiction and initialize GEN as the negation of a contradiction. On new data POS, disjoin the new positive example descriptions onto the old disjunction SPEC. On new data NEG, negate the result of removing the negation from the old disjunction GEN and disjoining on the new negative example descriptions in NEG. Each of these operations can be performed in time linear in the sizes of POS and NEG, and the overall version space maintenance procedure until the time of convergence operates in time polynomial in the total data received to that point. So boolean version space maintenance is just the computationally trivial matter of disjoining the data instances as they arrive. Of course, it is also no more interesting than memorizing POS and NEG.

But as it turns out, we have the following corollary to our proof that PLUfcCS is NP-complete:

Definition: Let the *boolean lattice upward closure complement search (BLUCCS* problem) be given as follows. Let U be a finite set and let P(U) be the power set of U. Let S be a subset of P(U) and let $K \leq |U|$. The problem is to decide whether there exists a $B \in P(U)$ such that $|B| \leq k$ and B is included in no element of P.

Corollary: BLUCCS is NP-compiete.

Proof:

We reduce MSC by the same strategy as before. Only now, the points of the boolean lattice automatically behave like boolean |U|-vectors so we can dispense with the AXIOMS and with doubling the size of the lattice. Q.E.D.

So how can a fast boolean version space maintenance procedure avoid solving this NP-complete problem? The reason is that the sets GEN and SPEC are always *unit subsets* of a finite boolean lattice (since GEN is the negated disjunction of the negative examples and SPEC is the disjunction of the positive examples). Hence, only trivial cases of the NP-complete problem need be solved by a solution to the boolean version space update problem. In a partition lattice of quantified concepts, however, this is not true, and the set SPEC may have a very ragged structure. Indeed, for each instance <n,k,S> of the PLUCCS problem that encodes an MSC problem instance, there is data POS such that S is a subset of SPEC, so each instance of the NP-complete problem MSC can appear in practice when updating version spaces of structural descriptions.

9. Applications to Refinement Operators in Theory Generation

In his Model Inference System for inferring true theories and PROLOG programs from facts, Ehud Shapiro invokes the notion of a *refinement operator* (Shapiro, 1981). For a more detailed discussion of refinement, see (Laird, 1985). Without belaboring details, a refinement operator is analogous to the more vaguely specified "generalization" operator in Mitchell's version space maintenance algorithm. That is, given a failed hypothesis, it generates a variety of logically weaker, more complex hypotheses to be tested. Those that pass are added to a set, just as in the version space case. The only difference is that the resulting set *is* the conjectured theory, rather than a set of mutually incompatible hypotheses as in the case of version spaces.

Many of Shapiro's proposed refinement operators ((Shapiro, 1981) pp. 23-32) have the property that for each hypothesis generated by the operator, the partition lattice of substitution instances of this hypothesis is embedded in the overall space searched by the operator. Since Shapiro's refinement operator search proceeds by splitting the graph of refinements into a search tree in the same way that Mitchell's version space maintenance algorithm does, one may expect this search to visit the same sentence intractably often. Indeed, a close inspection of the traces of Shapiro's Model inference system reveals that without various substantive, *a priori* restrictions on hypothesis syntax, an explosion of redundant formulations of the same axioms would arise.

10. Conclusion

Searches through the substitution variants of an hypothesis are ubiquitous in machine learning. They are encountered in testing hypotheses, in deciding entailment to eliminate redundant axioms or possibilities, and in searching refinement orderings for non-refuted hypotheses. In particular, these tasks arise in version space maintenance and in induction by refinement.

In this paper, I provided a logical analysis of version space maintenance and employed the theory of NP-completeness as a heuristic tool to focus on the difficulties that variable substitution can pose for standard inductive systems in artificial intelligence. The results presented should serve as fair warning to programmers who are pleased by learning programs that handle toy cases of entailment, consistency, or refinement search over structural descriptions. But more positively, these reductions may point to well-analyzed heuristic techniques known to provide fast, approximate solutions for the reduced problems (e.g. graph 3-colorability and minimal set cover. c.f. (Gusfield, 1987). Finally, results of this kind should assist in the transfer of information across different artificial intelligence communities (e.g. version spaces and induction by refinement) by highlighting strong formal analogies between different areas of research in artificial intelligence..

Acknowledgements

I thank Clark Glymour for his many useful suggestions concerning several early drafts of this paper. Without his generous encouragement I would not have written up its results. I would also like to thank Ken Manders for introducing me to the heuristic, exploratory value of the theory of NP completeness. Finally, I must thank R. Statman for useful hints concerning the reduction argument behind Theorem 1. Of course, all errors are my own fault.

References

Martin Algner. (1979). *Combinatorial Theory.* New York: Springer Verlag.

C. C. Chang and H. J. Keisler. (1973). *Model Theory.* Amsterdam: North-Holland.

Michael R. Garey and David S. Johnson. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: W.H. Freeman and Company.

Dan Gusfield and Leonard Pitt. (1987). Approximations for Subset Selection Problems.

David Haussler. (1987). Bias, Version Spaces and Valiant's Learning Framework. *Proceedings of the Fourth International Workshop on Learning.* Los Altos, Ca., Morgan Kaufmann.

M. Kearns, M. Li, L. Pitt, and L. Valiant. (1987). Recent Results on Boolean Concept Learning. *Proceedings of the Fourth International Workshop on Learning.* Los Altos, Ca., Morgan Kaufmann.

Kevin T. Kelly. (1986). *The Automated Discovery of Universal Theories.* Doctoral dissertation, University of Pittsburgh,

P.D.Laird. (1985). *Inductive Inference by Refinement* (Tech. Rep. YALEU/DCS/RR-376). Yale University Department of Computer Science,

Tom M. Mitchell. (1982). Generalization as Search. *Artificial Intelligence,* 75, 203-226.

T. M. Mitchell, Paul E. Utgoff, and Ranan Banerji. (1983). Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics, in Ryszard S. Michalski, Jaime G. Carbonell, Tom M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach.* Palo Alto, Ca.: Tioga Publishing Co.

Osherson, D. and Weinstein, S. (1986). Identification in the Limit of First Order Structures. *Journal of Philosophical Logic,* 75, 55-81.

L. Pitt and L.G. Valiant. (1986). *Computational Limitations on Learning from Examples* (Tech. Rep. TR-05-86). Center for Research in Computing Technology,

Ehud Y. Shapiro. (February 1981). *Inductive Inference of Theories from Facts* Research Report 1S2). Yale University: Department of Computer Science,

L J. Stockmeyer. (1973). Planar 3-Colorability is NP-Complete. *SIGACTNews,* 5.3, 19-25.

Patrick Henry Winston. (1975). Learning Structural Descriptions from Examples. In *The Psychology of Computer Vision.* New York: McGraw-Hill.

Notes

[1]Or at least nearly true, as Valiant (Kearns, 1987) and his followers prefer to require.

[2]A relational structure for L specifies a set D of objects called the domain of the structure, an nvary relation on D for each m-ary predicate of L, an nva*y total function on D for each nvary function symbol of L, and a distinguished individual for each individual constant of L. For details see (Chang, 1973).

[3]Conflatton of the learning problem with strategies for its solution is not unusual. Mitchell, for example speaks almost interchangably of following the consistency strategy and of solving one's concept learning problem. On page 204 he says that the "generalization problem* is "to infer the identity of some unknown target* generalization by observing a sample set of its training instances." (The term "infer" is undefined in the paper, and may be taken to range over the variety of convergence criteria reviewed above). But in the same paragraph he recharacterizes the problem as to determine "...generalizations within the provided language that are consistent with the presented training instances...." (Mitchell, 1982).

[4]This approach is suggested by Mitchell (p.216).

[5]The usual terminology is "more general than", but this is misleading since the defined extensions may be identical.

[6]e.g. in the case of concepts with polynomial definitions. For example, overweight(x) <~> p(weight(x),height(x))$\geq$k, where p is some polynomial function. Clearly, infinitely many extensionally inequivalent hypotheses of this form may be consistent with with the data, by fitting the polynomial to the finitely many data points observed so far.

[7]Mitchell defines version space procedures this way, but his published procedure (Mitchell, 1982) does not satisfy the definition. Instead of searching a fixed hypothesis space for GEN and for SPEC as data increases, Mitchell's procedure searches different hypothesis spaces depending (rather arbitrarily) on the number of distinct individuals mentioned in the first chunk of data presented. Specifically, if n individuals are mentioned in the first chunk of data, then no hypothesis in disjunctive normal form with more than n disjuncts is considered. I assume in this paper that the hypothesis space remain fixed on all data presentations.

[8]In the case of boolean concept learning, only the current chunk of data presented is required, and the

rest can be discarded. In such circumstances, the procedure is said to be run *incrementally.* The version space procedure does not work incrementally in general.

[9]Note that there is no reason to suppose that a unique definition is true, even up to logical equivalence.

[10]The literature on boolean version spaces tends to miss this fact. On the other hand, the LEX system (Mitchell, 1983) did make use of substantive axioms concerning inclusion relations among various syntactic properties of mathematical formulas in the integral calculus, and these were involved in the definition of the generalization and speicalization operators.

[11]For suppose it did. Let p be the assumed polynomial upper bound. Provide the data and run the generator until it makes an output or the time for generating the empty set is used up. If the time is used up, or nonempty sets are output, then GEN and SPEC are not empty and an hypothesis consistent with the data exists. If the empty set is output, then no such hypothesis exists.

[12]i.e. version spaces on an hypothesis language involving only single-place predicates and no quantifiers.

[13]Recall that the data consists of open formulas true under a fixed interpretation. Hence, each finite set of data can be thought of as the result of conjoining the literals in the set and placing existential quantifiers over all free variables occurring in the resulting conjunction.

[14]I am indebted to Rick Statman for a helpful conversation concerning this result.

[15]A clause is a universally quantified disjunction of literals.

[16]The expression involves summation over the elements of an integer partition.

[17]To justify the left hand side, note that for an arbitrary partition of cardinality k, there are (k,2) ways to select two cells from the k cells to identify. To see the identity, note that (k,2)=k(k-1)/2. Form the full product of terms of this form from (2,2) to (n,2). There are n-1 of them, which yields 2^{n-1} in the denominator. Grouping terms reveals n!(n-1)! in the numerator.

[18]Note that n!(n-1)! involves (n-3)+(n-3-1) = 2n-7 factors no smaller than 4. There is an n after which 2n-7 > n-1. So for each m greater than this n, each of the m-1 2's in the denominator of #paths(m) may be canceled against a factor in the numerator that yields a quotient no smaller than 2. So for each such

m, the result of the quotient is a product of $(n-i)+(n-2)$ « $2n-3$ factors no smaller than 2. So for each such m, #paths(m) > 2^{2n-3}.

[19]Note that this is not the same as *expected complexity,* in which the average is taken over the the problem instances of a given size. Here, the average is taken over the elements of the set generated, and we still maximize over this value with respect to problem instances of a given size.