In Proceedings AAAI-92

Polynomial-Time Learning with Version Spaces

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

Abstract

Although version spaces provide a useful conceptual tool for inductive concept learning, they often face severe computational difficulties when implemented. For example, the G set of traditional boundary-set implementations of version spaces can have size exponential in the amount of data for even the most simple conjunctive description languages [Haussler, 1988]. This paper presents a new representation for version spaces that is more general than the traditional boundary-set representation, yet has worst-case time complexity that is polynomial in the amount of data when used for learning from attribute-value data with tree-structured feature hierarchies (which includes languages like Haussler's). The central idea underlying this new representation is to maintain the traditional S boundary set as usual, but use a list N of negative data rather than keeping a G set as is typically done.

1. Introduction

Concept learning can be viewed as a problem of search [Simon and Lea, 1974; Mitchell, 1982]—to identify some concept definition out of a space of possible definitions expressible in a given concept description language. Mitchell [1982] formalized this view of *generalization as search* in his development of version spaces: A *version space* is the set of all classifiers expressible in the description language that correctly classify a set of data. Mitchell furthermore noted that the relative generality of concepts imposes a partial order that allows efficient representation of a version space by the boundary sets S and G containing the most specific and most general concept definitions in the space. The S and G sets delimit the set of all concept definitions consistent with the given data. Incremental learning is accomplished with the *candidate-elimination algorithm*, which manipulates only the boundary sets of a version space: as each new example is obtained, S and G are modified so that they represent a new version space containing those concept definitions that correctly classify all the previously processed data plus the new example.

Although version spaces have been successful as a conceptual framework for learning, they have had serious computational limitations as a way to implement learning systems. This paper addresses the issue of the computational complexity of implementing version-space learning algorithms by proposing an alternative representation of version spaces that maintains the S set as usual, but uses a list N of negative data rather than the traditional G set. The benefit of this new representation is that unlike the $[S, G]$ representation of version spaces where the G set can have size exponential in the amount of data even for very simple conjunctive languages [Haussler, 1988], this $[S, N]$ representation requires space and time only polynomial in the amount of data. It furthermore extends the version-space approach to situations where the boundary-set representation cannot be used.

The remainder of this paper is structured as follows. Section 2 discusses criteria for evaluating version-space representations, including various operations a representation should tractably support. Section 3 then presents the $[S, N]$ representation in more detail, including implementations of various operations using this representation. Descriptions of how these implementations can be instantiated for conjunctive languages over tree-structured attribute hierarchies to yield polynomial-time version-space algorithms follows in Section 4. Section 5 discusses some of the issues raised by this work, including the generality of the $[S, N]$ representation and some open problems, and Section 6 concludes the paper with a summary of the main points of this work.

This paper owes a great debt to William Cohen, in front of and with whom much of this work developed. The arrangement of facilities by Paul Rosenbloom at ISI (under DARPA and ONR contract number N00014-89-K-01555) so I could escape from New Jersey and work in the California sun is tremendously appreciated. Discussions with and comments from William Cohen, Steve Norton, and an onymous reviewer (Oren Etzioni) are also greatly appreciated.

2. Representing Version Spaces

The key idea in this paper is to develop an alternative representation of version spaces for which computational intractability does not occur. However, before describing the alternative representation, it is useful to elaborate a bit further on what is meant by a representation of a version space.

First some basic terminology. Concept learning problems traditionally assume the presence of some concept description language, *CDL*, in which results must be expressed. Each concept definition in *CDL* applies to a universe of objects, or instances, $\mathcal{I}$, some of which it classifies as positive (the concept definition is said to *cover* the instance) and some of which it classifies as negative. The *extension* of a concept description $c \in \mathit{CDL}$ is the set of all objects it classifies as positive. A learning problem is a triple $\langle \mathit{CDL}, I^+, I^- \rangle$, where $I^+, I^- \subseteq \mathcal{I}$. I^+ is the set of positive training data, and I^- is the set of negative training data. A concept definition $c \in \mathit{CDL}$ is consistent with I^+ and I^-, written $\mathit{cons}(c, I^+, I^-)$, if it covers every element of I^+ and doesn't cover any element of I^-.

The version-space approach to concept learning [Mitchell, 1982] is to determine everything in a concept description language that is consistent with I^+ and I^-. The *version space* of all elements of *CDL* consistent with I^+ and I^-, $\mathit{VS}_{\mathit{CDL}}(I^+, I^-)$, is equal to $\{c \in \mathit{CDL} \mid \mathit{cons}(c, I^+, I^-)\}$. When clear from context, the *CDL* subscript will be left out.

A *representation* of a version space for a class $\mathcal{C}$ of description languages is a finite data structure from which $\mathit{VS}_{\mathit{CDL}}(I^+, I^-)$ can be defined for all $\mathit{CDL} \in \mathcal{C}$. For example, the following are all representations of a version space:

1. The tuple $[S = \min(\mathit{VS}(I^+, I^-)), G = \max(\mathit{VS}(I^+, I^-))]$ is a representation of a version space for the class $\mathcal{C}$ of admissible languages,[1] since for admissible languages $\mathit{VS}(I^+, I^-) = \{c \in \mathit{CDL} \mid \exists s \in S, \exists g \in G, s \preceq c \preceq g\}$.
2. $\mathit{VS}(I^+, I^-)$, i.e., listing all elements in the version space, is trivially a representation of a version space for the class $\mathcal{C}$ of finite concept description languages.
3. The tuple $[I^+, I^-]$ is a representation of a version space for all languages, since $\mathit{VS}(I^+, I^-) = \{c \in \mathit{CDL} \mid \mathit{cons}(c, I^+, I^-)\}$.

While all of these are version-space representations, the key question that must be asked is how tractably a set of data can be processed for a particular representation. This can be formalized as follows. A version-space representation is *tractable* for a class of description languages if the representation of $\mathit{VS}(I^+, I^-)$ can be computed for all I^+ and I^- in time polynomial in $|I^+|$, $|I^-|$, and relevant properties of the description language (such as the number of features for feature-based description languages).[2]

For example, consider the tractability of the version-space representations discussed above:

1. The $[S, G]$ representation is not tractable for conjunctive languages of tree-structured features, since Haussler has shown that the G set can have size exponential in $|I^-|$ for even the simple subcase of monomial languages.
2. $\mathit{VS}(I^+, I^-)$ (explicitly listing the entire version space) is only tractable if the description language is small, i.e., it has size polynomial in $|I^+|$, $|I^-|$, and relevant properties of the description language.
3. The $[I^+, I^-]$ representation is tractable for conjunctive tree-structured languages since the $[S, N]$ representation of the version space can be computed from it, and as shown later this second representation is tractable.

In addition to processing a set of data, there are a number of operations that are useful when learning with version spaces, and a good representation should support as many of them as possible. This can be formalized as follows. A version-space representation is *epistemologically adequate* for a set of operations and a class of description languages if all the operations can be implemented using the representation. A representation is *heuristically adequate* for a set of operations and a class of description languages if each of the operations can be computed in time polynomial in the size of the operation's inputs and in relevant properties of the description language.

Among the operations that have generally proven useful for version space are the following:

1(a) *Collapsed?*$(\mathit{VS}) \rightarrow$ `true` or `false`:
If data are inconsistent or if there is no description that can distinguish between all given positive and negative data, the version space will be empty. This operation returns true if version space VS is empty, and otherwise it returns false.

(b) *Converged?*$(\mathit{VS}) \rightarrow$ `true` or `false`:
The ideal result of learning will be a version space that contains only a single item. This operation returns true if version space VS contains exactly one concept definition, otherwise it returns false.

[1] Admissibility is a property that must hold for a concept description language to guarantee that a version space can be represented by its boundary sets [Gunter *et al.*, 1991; Hirsh, 1991; Mitchell, 1978]

[2] Tractability should not be confused with the important question of *sample complexity* (such as is addressed in the pac-learning literature), namely how much data is necessary to learn well with a particular class of description languages. The results in this paper guarantee that if the sample complexity is polynomial in relevant features of the language—such as to ε-exhaust a version space with high probability [Haussler, 1988]—processing the data using the $[S, N]$ representation will have polynomial computational complexity; there are no such guarantees for the $[S, G]$ representation.

2(a) $\underline{\textit{Update}(I, \textit{VS}) \rightarrow \textit{VS}'}$:
For each new example learning must remove those elements of the version space that classify the example incorrectly. This operation updates version space VS to reflect new instance I.

(b) $\underline{\textit{Classify}(I, \textit{VS}) \rightarrow +, -, \text{or } ?}$:
If all elements of a version space agree on the classification of an example, the example can be given the unanimous classification accorded by the version space. This operation returns "+" if every element of VS classifies I as positive, returns a "−" if every element of VS classifies I as negative, and otherwise returns "?" since some elements of VS classify I as positive and some as negative.

3(a) $\underline{\textit{Member}(C, \textit{VS}) \rightarrow \texttt{true or false}}$:
A version space is simply a set of concept definitions. This operation checks if concept definition C is in version space VS.

4(a) $\underline{VS_1 \cap VS_2 \rightarrow VS'}$:
Version spaces are sets and thus can be intersected. This operation returns the version space VS' that is the intersection of VS_1 and VS_2.

(b) $\underline{VS_1 \cup VS_2 \rightarrow VS'}$:
As for intersections, version spaces can also be unioned together. This operation returns the version space VS' that is the union of VS_1 and VS_2.

(c) $\underline{VS_1 \subseteq VS_2,\ VS_1 = VS_2 \rightarrow \texttt{true or false}}$:
One set can always be a subset of another. Similarly, two sets can be equal. These operations return true if VS_1 is a subset of VS_2 or if VS_1 is equal to VS_2, respectively.

The operations in 1, 2, 3, and 4a were described by Mitchell [1978]. The operations in 4 have also proven useful for an alternative form of boundary-set-based version-space learning [Hirsh, 1990].

To make this tangible, consider the adequacy of the three representations discussed earlier:

1. The $[S, G]$ representation is epistemologically adequate for the set of operations listed above for the class of admissible description languages, and it is heuristically adequate for the operations above for conjunctive languages of tree-structured features.
2. $VS(I^+, I^-)$ (explicitly listing the entire version space) is epistemologically adequate for finite description languages, and is only heuristically adequate for small description languages.
3. The $[I^+, I^-]$ representation is epistemologically adequate for the above operations for admissible description languages, since at worst the $[S, N]$ representation can be computed from it. It is furthermore heuristically adequate for all operations but 4b by computing the associated $[S, N]$ representation.

Since $VS(I^+, I^-)$ can be (although in theory need not be) computed by sequentially processing the individual elements of I^+ and I^- with a series of *Update* operations, one might think that the heuristic adequacy of a representation for the *Update* operation implies tractability for processing a set of data within the representation. This is not the case; for example Haussler has shown that for the boundary-set representation each *Update* can potentially double the size of the G set, yielding a result with size exponential in $|I^-|$. Tractability *can* be established if the representation is heuristically adequate for *Update* and if for all I^+ and I^- the size of the representation of $VS(I^+, I^-)$ is guaranteed to be polynomial in $|I^+|$, $|I^-|$, and relevant properties of the description language.

This paper is not the first to consider alternative representations of version spaces. Most relevant to this work is the representation used by INBF [Smith and Rosenbloom, 1990]. It represents a version space using the standard S set and using a G set that is only updated with a subset of the negative data—those that are "near misses"—and instead also maintains a list of "far miss" negative data. They show that for conjunctive tree-structured languages their representation is epistemologically and heuristically adequate for operations 1a, 1b, and 2a, and that it is furthermore tractable.

Also related is the work of Idestam-Almquist [1989], whose primary focus is not on the tractability of version spaces, but rather on retracting data as a way to handle inconsistent data. Idestam-Almquist assumes conjunctive languages over ranges and tree-structured features. However, rather than representing S and G sets he instead maintains where the S and G set would be for each feature after processing each of the training examples in isolation. Idestam-Almquist shows that his representation is epistemologically and heuristically adequate for conjunctive languages over ranges and tree-structured hierarchies for the operations in categories 1 and 2 plus his instance retraction operation.

The representation described in this paper bears many similarities to that of Smith and Rosenbloom, but does away with the G set altogether, extending the approach to non-feature-based languages and further version-space operations. Some of Idestam-Almquist's implementations of the operations in categories 1 and 2 are special cases of the implementations to be described in the next section.

3. Version Spaces without G Sets

The whole point of this paper is that a version space can be represented as a tuple $[S,N]$, where S represents the S boundary set of the traditional candidate-elimination algorithm (i.e., $S = \min(VS(I^+, I^-))$), and N is a list of the observed negative data (i.e., $N = I^-$). Initially S contains the empty concept that says nothing is positive[3] and N is empty.

[3]This should not be confused with an empty S set—the S set here has one concept definition whose *extension* is empty [Hirsh, 1990].

First, observe that this is indeed a representation of version spaces for admissible description languages, since $VS(I^+, I^-) = \{c \in CDL \mid \exists s \in S, s \preceq c, cons(c, \emptyset, N)\}$.

The remainder of this section presents epistemological adequacy conditions for the $[S, N]$ representation. The following shows that the representation is epistemologically adequate for operations 1a, 2a&b, 3a, and 4a for admissible languages (including proof sketches for the correctness of the given algorithms):

1(a) *Collapsed?*($[S, N]$): If S is empty return true, otherwise return false. (Cf. the algorithm for Update.)

Proof: If S is empty the version space has no minimal element, which means it must be empty (since the *CDL* is admissible). □

2(a) *Update*($I, [S, N]$): If I is negative add I to N and remove those elements of S that cover I; if I is positive replace S with the minimal generalizations of S that cover I but cover no element n of N.

Proof: The proof for the S set parallels that for the candidate-elimination algorithm [Mitchell, 1978]; the proof for the N set is obvious. □

(b) *Classify*($I, [S, N]$): If I is below every s in S, return "+" (as in the standard boundary-set implementation of version spaces). Otherwise compute the new version space $[S', N]$ that would arise if I were positive. If S' is empty (i.e., the version space collapses), I must be negative so return "−". Otherwise return "?".

Proof: If I is below every element of S every element of the version space must classify it positive. If treating I as positive causes the version space to collapse every element of the version space must classify it as negative. Otherwise some classify it as positive and some as negative. □

3(a) *Member*($C, [S, N]$): If C is above some s in S and covers no n in N return true, otherwise return false.

Proof: If a definition is above some element of S and doesn't cover any element of N it is in the version space. □

4(a) $[S_1, N_1] \cap [S_2, N_2]$: Compute the minimal pairwise generalizations of elements from S_1 and S_2. Throw away those that cover some element of N_1 or N_2. The S set for the new version space contains the minimal elements of the result of this computation. The new N set is simply the union of N_1 and N_2.

Proof: The proof for the S set parallels that for version-space merging [Hirsh, 1990]; the proof for the N set is obvious. □

In addition, the $[S, N]$ representation is epistemologically adequate for *Converged?* for the following two languages:

- If learning uses a conjunctive tree-structured language, then Smith and Rosenbloom [1990] have shown that convergence requires one "near-miss" for each feature in the S set that has not been generalized to be a "don't care". Thus it is possible to check convergence for such languages by checking if N contains the necessary near misses.

  Proof: [Smith and Rosenbloom, 1990]. □

- If it is possible to generate all minimal generalizations of a concept definition (i.e., for a concept definition c it is possible to generate all terms c_i for which c_i is more general that c and there is no c' between c and c_i), then simply compute the set of minimal generalizations of the S set (if it is singleton) and if each minimal generalization covers some element n in N the version space has converged.

  Proof: If there is no more general term that excludes all negative data, the singleton element of the S set must be the most general term consistent with the data as well as the most specific, and thus it is the only thing in the version space. □

These algorithms cover all of Mitchell's original operations, and only leave operations 4b and 4c. Note that the $[S, N]$ representation is *not* epistemologically adequate for computing unions, since it is not even possible to represent arbitrary unions of version spaces for conjunctive tree-structured languages in the $[S, N]$ representation (generating an example that demonstrates this is left as an exercise due to space limitations). On the other hand, the $[S, N]$ representation is epistemologically (and heuristically) adequate for subset and equality testing for conjunctive tree-structured languages by exploiting the results of Smith and Rosenbloom [1990], but space considerations preclude presenting details.

4. Polynomial-Time Learning with Version Spaces

The previous section demonstrated the correctness of the $[S, N]$ representation of version spaces and showed that it is epistemologically adequate for all the given operations (except 4b) for fairly weakly constrained description languages. However, to be a satisfactory representation it must be both tractable and heuristically adequate. This section demonstrates the tractability and heuristic adequacy of the $[S, N]$ representation for conjunctive tree-structure languages.[4]

[4] Although not discussed here, the algorithms of this section also apply (with minor changes) more generally, such as for real-valued data where the description language takes the form of conjunctions of ranges over the various variables. However, characterizing the exact linguistic limits of tractability for these algorithms is beyond the scope of this paper.

The most important criterion for a version-space representation is whether data can be processed in polynomial time, i.e., whether the representation is tractable, and this is the first question considered here. As discussed earlier, tractability can be established by establishing heuristic adequacy for *Update* plus establishing polynomial bounds on the space requirements for representing $VS(I^+, I^-)$ for any I^+ and I^-. The heuristic adequacy of *Update* is shown below. It is easy to show the necessary space bounds by noting that the $[S, N]$ representation of $VS(I^+, I^-)$ (for the class of conjunctive tree-structured languages) requires space linear in only $|I^-|$ and the number of features, since the S set will always be singleton, and the size of N can never be larger than the amount of negative training data (in contrast to the traditional $[S, G]$ representation, which may require exponential space). Thus the $[S, N]$ representation is tractable.

The remainder of this section shows the heuristic adequacy of the $[S, N]$ representation for conjunctive tree-structured languages for all the operations in categories 1 through 3 plus operation 4a using the algorithms of the previous section. Key to these results is the fact that comparing two concept definitions for relative generality and computing the minimal generalization of two definitions requires time at worst linear in the number of features and the sizes of the generalization hierarchies, as does comparing an instance to a concept definition or computing the minimal generalization of a definition that covers an instance. The goal here, of course, is to show that each resulting implementation requires time at worst polynomial in the size of the inputs (e.g., $|N|$), the number of features, and the size of the feature hierarches.

1(a) *Collapsed?*($[S, N]$): This simply checks if S is empty, and is thus trivially tractable.

(b) *Converged?*($[S, N]$): For attribute-value data with generalization hierarchies the result of Smith and Rosenbloom [1990] mentioned above can be used. This simply requires scanning every element of N and checking if it is a near miss by inspecting S and the generalization hierarchies for each feature. This clearly takes time polynomial in $|N|$, the number of features, and the size of the generalization hierarchies.

2(a) *Update*($I, [S, N]$): For negative data this requires at worst checking if the single element of S covers I and adding I to N, which only requires time linear in the number of features and the hierarchy sizes. For positive data this requires computing the minimal generalization of S and I, then comparing the result to each element of N, which requires time at worst polynomial in $|N|$, the number of features, and the generalization-hierarchy sizes.

(b) *Classify*($I, [S, N]$): At worst this requires comparing I to S (requiring time linear in the number of features and the hierarchy sizes), then updating the version space with the example (which as just described requires polynomial time), then finally checking if the resulting S set is empty. This all requires polynomial time.

3(a) *Member*($C, [S, N]$): Checking whether C is above the singleton S-set element requires time linear in the number of features and the hierarchy sizes, and checking that it covers no element of N requires time polynomial in $|N|$, the number of features, and the hierarchy sizes, and thus membership takes at most polynomial time.

4(a) $[S_1, N_1] \cap [S_2, N_2]$: Computing the generalization of the singleton elements from S_1 and S_2 requires time polynomial in the number of features and the hierarchy sizes. Comparing it to all elements of N_1 and N_2 requires at most $|N_1| + |N_2|$ comparisons, and thus computing the new S set requires polynomial time. Computing the new N set simply requires unioning N_1 and N_2, which is of course tractable.

5. Discussion

This paper has described how the $[S, N]$ representation of a version space is epistemologically adequate for most of the desired version-space operations with few restrictions on description languages beyond admissibility. It also showed the tractability and heuristic adequacy of the $[S, N]$ representation when used in learning from attribute-value data with tree-structured feature hierarchies. This is particularly notable because it applies even in cases where the traditional boundary-set implementation of version spaces must store an exponential number of items in the G set, i.e., where the $[S, G]$ representation is not tractable. Since the approach described here does not maintain a G set this exponential complexity never occurs.

Eliminating the G set has further benefits beyond avoiding exponential G-set size. These include:

- If the G set is infinite. For example, if $\mathcal{I}$ is infinite and *CDL* contains one most general definition whose extension contains all of $\mathcal{I}$ and the remaining elements of *CDL* include all definitions that cover exactly one or exactly two elements of $\mathcal{I}$, after a single positive example and a different single negative example the G set would have infinite size. Nonetheless, the $[S, N]$ representation would still apply to this case. A more realistic example of this is in the description logic CLASSIC [Borgida *et al.*, 1989], where there are an infinite number of maximally general specializations of THING that would exclude a description containing a cyclic SAME-AS construct; here, too, the G set would be infinite.
- If the language is not admissible. Even if there are unbounded ascending chains (which means there may be no maximal term for a version space and hence no G set), the $[S, N]$ representation may apply; all that is necessary is that the S set be repre-

sentable. This suggests that the notion of admissibility can be separated into two parts: "lower" admissibility for S-set representability and "upper" admissibility for G-set representability. All the earlier proofs only require lower admissibility.

- If it is intractable or undecidable to compute a single minimal specialization of a concept definition. Since the $[S, N]$ representation only requires generalizing S-set elements, it applies even when G-set specializations cannot be computed.

Thus the $[S, N]$ representation of version spaces makes it possible to implement version-space learning in situations where Mitchell's original version-space approach would not have even applied.

This paper has suggested doing away with G sets since most languages traditionally used with version spaces tend to have singleton S sets and it is the G set that causes difficulties. However, for some languages, such as pure disjunctive languages, the G set remains singleton and the S set can grow exponentially large. While this paper has described representing version spaces by the S and N sets, by symmetry it is possible to represent a version space by its G set and a list P of positive data. In general one would want to represent a version space by whichever of the two boundary sets would remain small plus the data that correspond to the other boundary set.

There are a number of questions that remain open. This paper described how it is possible to do away with one boundary set by instead saving a list of training data. The obvious next step would be to replace both boundary sets with the lists of positive and negative data that give rise to them and perform all operations using these lists. While presented as a strawman representation in Section 2, the general questions of tractability and heuristic adequacy are worth exploring.

Although the $[S, N]$ representation is heuristically adequate for subset and equality testing for conjunctive languages over both tree-structured hierarchies and ranges, the general question of epistemological and heuristic adequacy for these operations for more general languages remains an open question. Lastly, the tests for convergence presented here only apply in special cases; while it would be interesting to show epistemological adequacy for a more general class of languages, the assumption that it is possible to compute all minimal generalizations of a definition is likely to be the most general case possible.

Finally, note that the $[S, N]$ representation is not a perfect replacement for the traditional $[S, G]$ boundary-set representation. If unions are important, or if it is necessary to have the G-set explicitly (such as if false negatives are bad and thus very general terms are desired), the $[S, G]$ representation is the only one known to be appropriate.

6. Summary

This paper has presented a new representation for version spaces that maintains the S set of traditional boundary-set implementations of version spaces, but replaces the G set with a list N of negative data. This $[S, N]$ representation was shown to support various desired version-space operations. When applied to attribute-value data with tree-structured feature hierarchies learning has worst-case complexity that is polynomial in the amount of data even in situations where the candidate-elimination algorithm takes exponential time and space. The learning algorithms also apply in cases where boundary-set implementations of version spaces cannot be used, such as when it is intractable or impossible to compute G-set elements.

References

A. Borgida, R. J. Brachman, D. L. McGuiness, and L. Resnick. CLASSIC: A structural data model for objects. In *Proceedings of SIGMOD-89*, Portland, Oregon, 1989.

C. A. Gunter, T.-H. Ngair, P. Panangaden, and D. Subramanian. The common order-theoretic structure of version spaces and ATMS's (extended abstract). In *Proceedings of the National Conference on Artificial Intelligence*, pages 500–505, Anaheim, CA, July 1991.

D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 26(2):177–221, Sept. 1988.

H. Hirsh. Incremental version-space merging. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 330–338, Austin, Texas, June 1990.

H. Hirsh. Theoretical underpinnings of version spaces. In *Proceedings of the Twelfth Joint International Conference on Artificial Intelligence*, pages 665–670, Sydney, Australia, August 1991.

P. Idestam-Almquist. Demand networks: An alternative representation of version spaces. SYSLAB Report 75, Department of Computer and Systems Sciences, The Royal Institue of Technology and Stockholm University, 1989.

T. M. Mitchell. *Version Spaces: An Approach to Concept Learning.* PhD thesis, Stanford University, December 1978.

T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, March 1982.

H. Simon and G. Lea. Problem solving and rule induction. In H. Simon, editor, *Models of Thought.* Yale University Press, 1974.

B. D. Smith and P. S. Rosenbloom. Incremental nonbacktracking focusing: A polynomially bounded generalization algorithm for version spaces. In *Proceedings of the National Conference on Artificial Intelligence*, pages 848–853, Boston, MA, August 1990.