Incremental Non-Backtracking Focusing: A Polynomially Bounded Generalization Algorithm for Version Spaces *

Benjamin D. Smith
University of Southern California
Department of Computer Science
Los Angeles, CA 90089-0782
benjamin@castor.usc.edu

Paul S. Rosenbloom
Information Sciences Institute
University of Southern California
Marina Del Ray, CA 90292
rosenbloom@vaxa.isi.edu

Abstract

The candidate elimination algorithm for inductive learning with version spaces can require both exponential time and space. This article describes the Incremental Non-Backtracking Focusing (INBF) algorithm which learns strictly tree-structured concepts in polynomial space and time. Specifically, it learns in time $O(pnk)$ and space $O(nk)$ where p is the number of positives, n the number of negatives, and k the number of features. INBF is an extension of an existing batch algorithm, Avoidance Focusing (AF). Although AF also learns in polynomial time, it assumes a convergent set of positive examples, and handles additional examples inefficiently; INBF has neither of these restrictions. Both the AF and INBF algorithms assume that the positive examples plus the near misses will be sufficient for convergence if the initial set of examples is convergent. This article formally proves that for tree-structured concepts this assumption does in fact hold.

Introduction

The candidate elimination (CE) algorithm [Mitchell, 1982] learns a concept from a set of positive and negative examples of that concept. The concept to be learned is called the *target concept*. A concept is usually described by a tuple of *features* each of which can have one of several values. These values can be discrete, tree-structured, or lattice-structured. The features and their possible values comprise the generalization language for a particular domain.

The CE algorithm learns the target concept by searching a *hypothesis space* consisting of all concepts described by the generalization language. The concepts are organized into a lattice defining a partial order of generality on the concepts. This space is searched bidirectionally for the target concept. The top-down search maintains a set, G, of the maximally general concepts in the space which are still candidates for the target concept. A negative example causes all concepts in G which cover it to be specialized just enough to exclude the example. A positive example prunes concepts in G which do not cover it. The bottom-up search maintains a set, S, of maximally specific candidates. A positive example generalizes concepts in S which do not include it just enough to include the example. A negative example prunes concepts from S which do not cover it. Initially, S contains the first positive example and G contains the maximally general concept in the space. The examples are then processed incrementally until convergence ($S = G$). At this point, there is only one concept remaining in S ∪ G which is consistent with the examples (assuming a conjunctive concept language). This is the target concept, C.

That the CE algorithm can consume space exponential in the number of negative examples has been shown in [Haussler, 1988]. This rapid growth is caused by *fragmentation* of the G set. When G is specialized by a common class of negative examples known as *far misses* (described later), each concept in G is specialized into several concepts (always the same number for a given far miss). Each of these specializations appears in the specialization of G, so a given far miss can be thought of as fragmenting each concept in G into several concepts.

Fragmentation means G can grow exponentially in the number of far misses. Since the candidate elimination algorithm requires examination of every concept in G for each new example, time is also exponential. When learning concepts with all tree-structured features, S contains exactly one concept [Bundy *et al.*, 1985] and thus does not impact the time and space bounds. So for such concepts, if G could be kept from fragmenting, then the time and space bounds could possibly be brought into polynomial ranges in the number of examples. This is the central principle behind the Incremental Non-Backtracking Focusing (INBF) algorithm.

The INBF algorithm is an extension of the avoidance

*This project is partially supported by the National Aeronautics and Space Administration under cooperative agreement number NCC 2-538. Thanks go to Haym Hirsh, Yoram Reich, and Richard Young for their helpful comments on earlier drafts of this article.

focusing (AF) algorithm [Bundy *et al.*, 1985]. In general, focusing algorithms for tree- or lattice-structured concepts [Young and Plotkin, 1977] bound the search space with two concepts, upper and lower, instead of with two sets, S and G. For a negative example, upper is specialized on one feature to exclude the value of the example on that feature. If this feature's value is not included by the target concept, then this is a correct specialization, otherwise the specialization leads to incorrect concept formation. When the mistake is discovered, the focusing algorithm backtracks and selects a different feature on which to specialize upper. Backtracking is focusing's equivalent of fragmentation.

The AF algorithm [Young and Plotkin, 1977; Bundy *et al.*, 1985] is a batch algorithm that avoids backtracking by assuming that it has been given a convergent set of examples, and processing all the positive examples first so that lower is at the target concept, and then processing just the near misses[Winston, 1975] to converge upper to the target concept. This assumes that the near misses will always be sufficient for convergence. Converging upper to the target concept confirms that the examples are consistent. A near miss is an example for which only one feature is not covered by the target concept. Since lower is at the target concept, all the near misses can be correctly identified along with the feature that is not covered by the target concept. Since this feature is known for each near miss, choosing this feature always leads to a correct specialization, and so the AF algorithm never backtracks.

INBF uses a similar approach to avoid backtracking (fragmentation). However, there are some inherent drawbacks to the AF algorithm which INBF avoids. AF assumes that there are enough positive examples for lower to converge to the target concept. If there are insufficient positive examples, so that lower is below C, then correct identification of all the near misses can not be guaranteed. Furthermore, if more examples are presented later, they may generalize lower, in which case the original negatives must be reprocessed since they may now be identifiable as near misses. But because AF is a batch algorithm, reprocessing the old negatives is very inefficient, especially with many new example sets. Lastly, many induction problems require an incremental algorithm, not a batch algorithm. Though AF could be rerun with the new examples, it would be terribly inefficient, even when modified to save as much work as possible from previous batches.

This article introduces the INBF algorithm, an incremental focusing algorithm which has tri-polynomial space and time bounds in the number of positive examples, negative examples, and features. This algorithm is limited to learning tree-structured concepts[1] in conjunctive languages. Being incremental, additional examples are handled efficiently, and it uses less space than its batch counterpart. Unlike the AF algorithm, the INBF algorithm (with extensions) can learn from an incomplete set of positive examples. In this case, the concept learned will be incomplete, but equivalent to the one learned by the CE algorithm from the same set of examples.

The rest of this article first describes the fragmentation problem, then explains the INBF algorithm and how it avoids this problem. This is followed by time and space comparisons of the INBF, AF, and CE algorithms. Next, it is proven that the near misses plus the positive examples are indeed sufficient for convergence if the entire set of examples is convergent, as is required by both the AF and INBF algorithms. Though several authors have suggested that this is the case [Young and Plotkin, 1977; Bundy *et al.*, 1985; Hirsh, 1990], no formal proof appears in the literature. Finally, an extension to INBF is suggested that handles non-convergent examples, and limitations of the INBF algorithm are discussed.

Fragmentation

When G is specialized by a negative example, the specialization must cover the target concept but not the negative example. Only specializatons of G made on certain features, called *guilty features*, will satisfy these criterion. Specifically, guilty features are exactly those features, f, for which feature f of the negative example is not covered by feature f of the target concept. Fragmentation occurs because it is not generally possible to distinguish the guilty features of a negative example from the non-guilty features. At best, only some of the non-guilty features can be identified. These are *exhonerated features*. The non-exhonerated features include all of the guilty features and some non-guilty features. This is the set of *possibly guilty features*. Since the CE algorithm can not accurately select a single guilty feature from this set, it specializes G on *all* of the features in the set. Specializations made from non-guilty features will not cover the target concept and will eventually be pruned by positive examples. Only specializations made from the guilty features will remain. Each possibly guilty feature makes its own specialization of each concept in G. So if there are n possibly guilty features, then for every concept in G there are n concepts in the specialization of G. G has *fragmented*.

But why can't the guilty features be distinguished from the others? Guilty features are those features of a negative example which are not covered by the target concept. But the target concept is not known *a priori*, so the guilty features can not always be distinguished from the others. Some features, though, can be exhonerated as definitely not guilty. All concepts subsumed by S are subsumed by the target concept, so if feature f of the negative example is subsumed by feature f of a concept in S, then f can not be guilty. How-

[1] Discussion with Richard Young leads us to believe that INBF should be extensible to upper semi-lattices, as described in [Young and Plotkin, 1977].

ever, some concepts subsumed by the target concept are not subsumed by S, so some non-guilty features can not be exhonerated. Since all of the non-guilty features can not be exhonerated, the guilty features can not be distinguished from the non-guilty features. The more general S is, the more concepts it covers and the more non-guilty features it can exhonerate. When S reaches the target concept, all of the non-guilty features can be exhonerated.

The only kind of negative example guaranteed not to fragment G is a *near miss*. These examples have only one possibly guilty feature, so each concept in G maps to only one concept in G's specialization. By contrast, a negative example with more than one possibly guilty feature is called a *far miss*. The more specific S is, the fewer non-guilty features it can exhonerate, and the more features it must therefore label as possibly guilty. Thus a negative example which has several possibly guilty features (a far miss) with respect to a specific S may have only one possibly guilty feature (be a near miss) with respect to a more general S. Since S increases monotonically in generality to the target concept, examples which are initially classifiable only as far misses may later be identifiable as near misses. The INBF algorithm relies on this phenomenon.

The Algorithm

The INBF algorithm is an incremental version of the batch AF algorithm. Like the AF algorithm, the INBF algorithm exploits the fact that upper is guaranteed to converge without backtracking to the target concept when specialized only by all of the near misses from a convergent set of examples. The key is guaranteeing that *all* of the near misses can be recognized, and therefore used. But a near miss could possibly be classified as a far miss if lower is more specific than C. The batch algorithm guarantees identification of all near misses by processing all of the positive examples first so that lower is at C (assuming a convergent set of examples). Then just the near miss negatives are processed, and the far misses are discarded.

An incremental algorithm, however, can not expect all the positive examples first, so lower is more specific than C during most of the learning. Therefore INBF can not simply discard far misses as does AF: near misses that should be processed might be discarded because they were temporarily misclassified as far misses. Since these far misses may later be recognizable as near misses as lower converges to C, INBF saves the far misses on a list until they can be recognized. Every time lower is generalized, the list of saved far misses is scanned for far misses which are now recognizable as near misses. These near misses are used to specialize upper, and are then removed from the list.

Figure 1 presents the INBF algorithm. lower is initialized to the first positive example, p^1; upper is initialized to the maximally general concept, (ANY, ..., ANY);

```
INBF()
  lower= p^1; upper = (ANY, ..., ANY); wait-list= {}
  LOOP until upper = lower OR collapse
    For a positive example, p
      Generalize lower to include p
      FOR each negative example, n, in wait-list DO
        PROCESS-NEGATIVE(n)
    For a negative example, n
      IF n is not covered by (less than) upper
      THEN Discard n
      ELSE PROCESS-NEGATIVE(n)

PROCESS-NEGATIVE(n)
  Compare n and lower to determine number of
  possibly guilty features.
  CASE (number of possibly guilty features) OF
      guilty = 0 : collapse
      guilty = 1 : Specialize upper on the single
                   guilty feature of n.
                   Remove n from wait-list.
      guilty > 1 : Add n to wait-list if it's
                   not already there.
```

Figure 1: Incremental Non-Backtracking Focusing

and the list of saved negative examples, wait-list, is initialized to empty. If the algorithm receives a positive example, lower is generalized to include it. Some far misses may be recognizable as near misses by the new lower, so wait-list is scanned for near misses. upper is specialized to exclude any newly discovered near misses, and the near misses are removed from the list of saved negatives. If after an example has been processed upper and lower are equal, then they have converged to the target concept, and the algorithm has succeeded. Otherwise, the next example is processed.

Negative examples are first compared to upper. Those not less than upper would be classified as negative anyway, and are discarded. The CE algorithm performs a similar test. Otherwise, the negative is compared to lower to determine the number of possibly guilty features. If the example is guilty in zero features, then the version space is inconsistent. If the example is a near miss then it specializes upper. If the example is a far miss, then it is saved on wait-list until it can be recognized as a near miss.

Time/Space Summary

The INBF algorithm uses considerably less time and space than the CE algorithm in the worst case, and the same amount in the best case. The worst case bounds are derived as follows. Let p be the number of positive examples, n the number of negatives, and k the number of features. Each concept requires $O(k)$ units

of storage (for k features). INBF maintains only three variables: upper, lower, and wait-list. upper and lower are single concepts, and wait-list clearly has an upper bound of n concepts. Thus the total space requirement is $O(k) + O(k) + O(nk) = O(nk)$. PROCESS-NEGATIVE can be done in $O(k)$ time if its argument, n, a negative example, can be added or deleted from wait-list in constant time, and if the guilt of a feature of n can be determined in constant time. The latter is possible if examples are represented in vector notation, explained in the next section. Additions can be done in constant time by adding to the front of the list. For deletions, INBF knows where n is in wait-list at the time INBF calls PROCESS-NEGATIVE. If we pass this pointer to PROCESS-NEGATIVE along with n, then n can be deleted from wait-list in constant time. Each positive example calls PROCESS-NEGATIVE once for each example in wait-list. Each negative example makes a single call. Total time is thus $O(pnk) + O(nk) = O(pnk)$. Derivation of the other results appear in [Smith and Rosenbloom, 1990].

Worst Case Analysis			
	INBF	AF	CE
Time	$O(pnk)$	$O(pk + nk)$	$O(k^{n+1})$
Space	$O(nk)$	$O(pk + nk)$	$O(k^{n+1})$

The AF algorithm performs poorly on additional batches. All the negatives seen so far must be reprocessed for each new batch in order to find near misses not recognized in previous batches. The following is based on b batches of one positive and one negative example each.

Worst Case Analysis: Multiple Batches			
	INBF	AF	CE
Time	$O(b^2k)$	$O(b^3k)$	$O(k^{b+1})$
Space	$O(bk)$	$O(bk)$	$O(k^{b+1})$

The best case for all the algorithms occurs when all the negative examples are near misses. For the INBF algorithm, the list of saved negatives is always empty. For the CE algorithm, G never fragments.

Best Case Analysis			
	INBF	AF	CE
Time	$O(pk + nk)$	$O(pk + nk)$	$O(pk + nk)$
Space	$O(k)$	$O(pk + nk)$	$O(k)$

Near Miss Convergence Theorem

The Near Miss Convergence Theorem is the crucial result upon which both the INBF and AF algorithms rely. That is, that the positive examples plus the near misses alone are sufficient for convergence if the entire set of examples (positives plus negatives) are convergent. The theorem and proof are given formally below. The proof relies on a vector notation which is explained first, followed immediately by the proof of the theorem.

$P = \{e \mid e \text{ is a positive example}\}$
$F = \{e \mid e \text{ is a far miss negative example}\}$
$M = \{e \mid e \text{ is a near miss negative example}\}$
$N = \{e \mid e \text{ is a negative example}\} = F \cup M$

Theorem 1 (Near Miss Convergence Theorem) *If a set of strictly tree-structured examples $P \cup F \cup M$ of a concept is sufficient for convergence ($G = S = C$) of the Candidate Elimination Algorithm, then $P \cup M$ is also sufficient for convergence.*

Vector Notation

Concepts and examples with tree-structured attributes can be represented as vectors. This notation is used throughout the proof to simplify the arguments. Each feature of a concept (or example) is a component of the vector and is assigned an integer indicating its distance from p^1, the first positive example. The distance between feature i of a concept and feature i of p^1 is the number of nodes that feature i of p^1 must be moved up in the feature tree to reach a value which subsumes feature i of the concept.

For example, assume we have a concept with two features, each feature having the structure shown in Figure 2. If p^1 = (d,e), then the vector for (b,b) would be

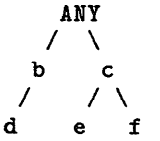

Figure 2: Feature Tree

(1,2). Feature i of vector p is denoted p_i. For any given tree-structured feature, i, the values that could be part of the target concept must lie between p_i^1 and ANY. This constrains the values to a single path from ANY to p_i^1. Thus, vector notation indicates, for a particular value, where that value lies on the path. Values off of the path are never part of the target concept, so each vector defines a unique concept. For example, if p_i^1 = e, then the only possible values for C_i are e, c, and ANY. 0 refers to e, 1 to c, and 2 to ANY.

If $A_i \geq B_i$ then A subsumes B on feature i. If $\forall i A_i \geq B_i$ then concept A subsumes concept B ($A \geq B$). Vectors form a partial order of generality, with smaller vectors being more specific than larger vectors.

Concept A can be generalized to include concept B by setting each feature, i, of A to $\max(A_i, B_i)$. Then $\forall_i A_i \geq B_i$, so A subsumes B. Furthermore A is the maximally specific concept which does so. A can be specialized to exclude B by causing A to not subsume B on some feature, i. This is done by setting A_i to $B_i - 1$. Since $B_i - 1$ is the smallest integer which fails to subsume B_i, A is the maximally general concept which

excludes B on feature i. Aside from notational simplicity, this method also makes specialization and generalization very fast operations. The usual tree searches can be avoided altogether once a concept's vector has been established with respect to p^1.

Definitions, Lemmas, and Observations

Definition 1 (Least Upper Bound of G: $\sqcup G$) *The Least Upper Bound of G, written $\sqcup G$, is the vector $(\sqcup G_1, \sqcup G_2, \ldots, \sqcup G_k)$, where each $\sqcup G_i$ is the most specific value for feature i s.t. $\forall g \in G \; \sqcup G_i \geq g_i$.*

Observation 1 *For tree-structured features, $\sqcup G_i$ can also be defined as the least common parent of the set of values taken by concepts in G on feature i. That is, $\sqcup G_i$ is the least common parent of the values in the set $\{g_i | g \in G\}$ where i is a tree structured feature.*

Observation 2 *Let V be a set of tree structured values. Let the least common parent of these values be denoted $\sqcup V$ (by Observation 1, the least common parent of V is also the least upper bound of V, hence the notation). Adding or deleting a value, v, from V will have no effect on $\sqcup V$ as long as v is subsumed by some other value in V. Formally, $\sqcup V$ is the least common parent of the following sets of tree structured values: V, $V \cup \{v\}$, and $V - \{v\}$ where v is subsumed by a value in V.*

Lemma 1 *In an uncollapsed version space with strictly tree-structured attributes $G = S$ iff $\{\sqcup G\} = S$.*

Lemma 2 *If G' is the set of concepts that result from specializing a set of tree-structured concepts, G, with a consistent far miss negative example, then $\sqcup G' = \sqcup G$.*

Observation 3 *If a set, G, contains only one element, x, then $\sqcup G = x$.*

Proof of Theorem 1

We must show that $P \cup M$ yields $G = S = C$. Assume that all the positive examples have been seen, and thus that $S = C$. By Lemma 1 it is therefore sufficient to show $\sqcup G = C$. But by Lemma 2 we know that F has no effect on $\sqcup G$, and thus whether or not $G = C$ is independent of F. Therefore since $P \cup M \cup F$ is sufficient for $\sqcup G = C$, so must be $P \cup M$. □

Proof of Lemma 1

(1) If $G = S$ then $\{\sqcup G\} = S$
Assume that $G = S$ and that the concepts in the version space have only tree-structured features. For version spaces with such concepts, S has only one element [Bundy *et al.*, 1985]. Call this element s. $G = S$ by assumption, so $G = \{s\}$. By Observation 3, $\sqcup G = s$ and thus $\{\sqcup G\} = \{s\} = S$.

(2) If $\{\sqcup G\} = S$ then $G = S$
Assume that $\{\sqcup G\} = S$, and that concepts in the version space have only tree-structured features. Then G has either one, or multiple elements (in a consistent version space, G can not have zero elements). Assume G has multiple elements. By Definition 1, $\forall g \in G \; g \leq \sqcup G$. But $\{\sqcup G\} = S$ by assumption, and since the version space only contains concepts with tree-structured features, S has only one element, s. Thus $\forall g \in G \; g \leq s$. But the version space collapses when every concept in G is strictly less than some concept in S, so there must be at least one element, g, of G that is equal to s. Since s subsumes every concept in G, so does g. But G contains only concepts which are not subsumed by other concepts in G, so $G = \{g\}$. This violates our assumption that G has multiple elements. Therefore G must have exactly one element. Call it x. Then by Observation 3, $\sqcup G = x$; but $\{\sqcup G\} = S$ by assumption, and thus $S = \{\sqcup G\} = \{x\} = G$. So $G = S$. □

Proof of Lemma 2

Let f be a consistent far miss, and G' be the set of concepts that results from specializing G with f. G' can be constructed from G and f by the construction algorithm (Figure 3) in two stages. Call the sets formed in these stages G^1 and G^2 respectively. The construction algorithm is the vector-notation equivalent of the action taken by the CE algorithm on a negative example. G^2 is then returned as G'. For each feature, i, let V_i be the set of values for feature i held by concepts in G, i.e., $V_i = \{g_i | g \in G\}$. Similarly, $V_i^1 = \{g_i^1 | g^1 \in G^1\}$ and $V_i^2 = \{g_i^2 | g^2 \in G^2\}$. We will show that for any feature, the set of values for the feature held by the concepts in each stage is formed by adding or deleting zero or more values which are subsumed by values in the previous stage. Thus by repeated application of Observation 2, $\sqcup V_i = \sqcup V_i^1 = \sqcup V_i^2$. Then by Observation 1, $\sqcup G_i = \sqcup V_i$ and $\sqcup G_i' = \sqcup V_i^2$. Thus $\sqcup G_i = \sqcup G_i'$, and since this holds for all i, $\sqcup G = \sqcup G'$.

First we show that for every feature, i, $V_i^1 = V_i \cup E_i$ where E_i is a set of values subsumed by values in V_i. In stage one, G^1 is initially empty. Every concept g in G then adds one concept to G^1 for every possibly guilty feature of f. When a given concept g in G, is specialized on a possibly guilty feature j, g adds one concept to G^1 with a value for feature j of $\min(g_j, f_j - 1)$. When specialized on a possibly guilty feature $k \neq j$, each g in G adds a concept to G^1 with a value of g_j for feature j. So if there are at least two possibly guilty

(1) FOR each possibly guilty feature, i, of $f \in F$ DO
    FOR each concept g in G DO
        $g' = g$
        $g_i' = \min(g_i, f_i - 1)$
        Add g' to G'
(2) Remove from G' every concept subsumed by another concept in G'.

Figure 3: Construction Algorithm

features, then for every feature i, the set of values, V_i^1, taken by the concepts in G^1 on feature i is $\{g_i | g \in G\} \cup \{\min(g_i, f_i - 1) | g \in G\} = V_i \cup E_i$. If i is not a possibly guilty feature, $E_i = \{\}$. We are guaranteed at least two possibly guilty features since G is being specialized by a far miss, f, and far misses always have multiple possibly guilty features. Note that for every i, $\min(g_i, f_i - 1) \leq g_i$, i.e, $\min(g_i, f_i - 1)$ is subsumed by g_i. Thus for every i, every value in E_i is subsumed by some value in V_i, so by Observation 2, $\sqcup V_i = \sqcup V_i^1$.

Next we show that V_i^2 is $V_i^1 - R_i$ where R_i is a set of values subsumed by values in V_i^1, and i is any feature. In stage two, G^2 is formed by removing from G^1 all concepts which are subsumed by another concept in G^1. Let the set of removed concepts be R. Thus V_i^2 is V_i^1 minus the set of values, R_i taken by concepts in R on feature i. But every concept in R is subsumed by a concept in G^1, so every value in R_i is subsumed by some value in V_i^1. Therefore $V_i^2 = V_i^1 - R_i$, and R_i is subsumed by V_i^1, so by Observation 2, $\sqcup V_i^1 = \sqcup V_i^2$. □

Incompletely Learned Concepts

There are many learning domains in which the availability of examples is limited, or for which prohibitively many examples are required for convergence. In such domains there are insufficient examples for convergence, so the set of examples is non-convergent and the target concept is incompletely learned. However, we would still like to classify new instances as being positive or negative examples of the incompletely learned target concept. In the CE algorithm, examples subsumed by S are always positive, those not subsumed by G are negative, and those between S and G can not be classified.

In the INBF algorithm, examples below lower are positive, and those not below upper are negative. But there could be examples under upper which the CE algorithm would classify as negative. This is because not all the near misses have been recognized and thus upper bounds the target concept more loosely than does G.

Let G-final be the G set after the CE algorithm has processed the (incomplete) set of examples. INBF can generate G-final from upper by processing all the saved negatives the same way the CE algorithm does. In the CE algorithm, G converges on G-final non-monotonically. Negative examples cause G to increase in size by fragmenting it, and positive examples reduce G by pruning concepts. Under many example orderings, G can initially fragment beyond the size of G-final, meaning that the size of G fluctuates as it approaches G-final. By contrast, the size of the G set in the INBF algorithm increases *monotonically* to the size of G-final. This is because only the saved *negative* examples are being processed to specialize upper. A negative example can only cause G to increase in size, so G must increase monotonically from one concept (upper) to several (G-final). Thus the maximum size attained by G for INBF is usually less, and never greater, than the maximum size of G for the CE algorithm. The degree to which the maximum sizes differ depends on the order of the examples.

Since the space bound of both algorithms is proportional to the maximum size attained by G, INBF saves space over the CE algorithm for most example orderings. For the remaining orderings, both algorithms have equivalent space bounds. Since time is proportional to space, these savings apply to the running time as well.

Conclusions and Limitations

For strictly tree-structured concepts, INBF is provably equivalent to the CE algorithm, but is polynomial rather than exponential. Unlike the AF algorithm, INBF can efficiently process additional examples, and can deal with non-convergent sets of examples and incompletely learned concepts. For such sets, INBF has the same worst-case time and space bounds as the CE algorithm, but better best-case bounds.

The INBF algorithm is restricted to learning strictly tree-structured concepts (and possibly upper semi-lattices). It is, at present, unable to learn concepts with arbitrary lattice-structured features. Lattice structured features allow pairs of values to have more than one least common parent. This means that there is more than one way to generalize a concept to include a positive example, and thus S can fragment. It also means that a near miss no longer specifies a unique specialization (generalization) of G (S), and thus near misses can not prevent fragmentation. Learning lattice-structured concepts is an area of future research.

References

(Bundy *et al.*, 1985) A. Bundy, B. Silver, and D. Plummer. An analytical comparison of some rule-learning programs. *Artificial Intelligence*, 27, 1985.

(Haussler, 1988) David Haussler. Quantifying inductive bias: Artificial intelligence learning algorithms and valiant's learning framework. *Artificial Intelligence*, 36, 1988.

(Hirsh, 1990) Haym Hirsh, 1990. Private communication.

(Mitchell, 1982) Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18, 1982.

(Smith and Rosenbloom, 1990) B.D. Smith and P.S. Rosenbloom. Polynomially bounded generalization algorithms for version spaces. In preparation, 1990.

(Winston, 1975) P. Winston. Learning structural descriptions from examples. In *The Psychology of Computer Vision*. McGraw-Hill, New York, 1975.

(Young and Plotkin, 1977) Richard M. Young and Gordon D. Plotkin. Analysis of an extended concept-learning task. In *IJCAI*, 1977.