Generalization as Search

Tom M. Mitchell

Computer Science Department, Rutgers University, New Brunswick, NJ 08903, U.S.A.

Recommended by Nils Nilsson

ABSTRACT

The problem of concept learning, or forming a general description of a class of objects given a set of examples and non-examples, is viewed here as a search problem. Existing programs that generalize from examples are characterized in terms of the classes of search strategies that they employ. Several classes of search strategies are then analyzed and compared in terms of their relative capabilities and computational complexities.

1. Introduction

'Learning' is a broad term covering a wide range of processes. We learn (memorize) multiplication tables, learn (discover how) to walk, learn (build up an understanding of, then an ability to synthesize) languages. Many subtasks and capabilities are involved in these various kinds of learning.

One capability central to many kinds of learning is the ability to *generalize*: to take into account a large number of specific observations, then to extract and retain the important common features that characterize classes of these observations. This generalization problem has received considerable attention for two decades in the fields of Artificial Intelligence, Psychology, and Pattern Recognition (e.g. [2–4, 6–10, 14, 15, 19–21]). The results so far have been tantalizing: partially successful generalization programs have been written for problems ranging from learning fragments of spoken English to learning rules of chemical spectroscopy. But comparing alternative strategies and developing general understanding of techniques has been difficult because of differences in data representations, terminology, and problem characteristics.

The purpose of this paper is to compare various approaches to generalization in terms of a single framework. Toward this end, generalization is cast as a search problem, and alternative methods for generalization are characterized in terms of the search strategies that they employ. This characterization uncovers similarities among approaches and leads to a comparison of relative capabilities

Artificial Intelligence **18** (1982) 203–226

0004-3702/82/0000–0000/$02.75

and computational complexities of alternative approaches. The characterization allows a precise comparison of systems that utilize different representations for learned generalizations.

2. The Problem

The class of generalization problems considered here can be described as follows: A program accepts input observations (instances) represented in some language, which we shall call the *instance language*. Learned generalizations correspond to sets of these instances and are formulated by the program as statements in a second language, which we shall call the *generalization language*. In order to associate instances with generalizations, the program must possess a *matching predicate* that tests whether a given instance and generalization match (i.e., whether the given instance is contained in the instance set corresponding to the given generalization).

Given the instance language, generalization language and matching predicate, the generalization problem is to infer the identity of some unknown 'target' generalization by observing a sample set of its training instances. Each *training instance* is an instance from the given language, *along with* its classification as either an instance of the target generalization (positive instance) or not an instance of the target generalization (negative instance). This generalization problem can be summarized as follows.

Generalization problem:

Given: (1) A language in which to describe instances.
(2) A language in which to describe generalizations.
(3) A matching predicate that matches generalizations to instances.
(4) A set of positive and negative training instances of a target generalization to be learned.

Determine: Generalizations within the provided language that are consistent with the presented training instances (i.e., plausible descriptions of the target generalization).

Here, a generalization is considered to be *consistent* with a set of training instances if and only if it matches every positive instance and no negative instance in the set. With this strict definition of consistency we assume (1) that the training instances contain no errors and (2) that it is possible to formulate a correct description of the target generalization within the given generalization language. Although several of the systems discussed in this paper have attempted to deal with learning from inconsistent training data, an analysis of performance in such cases is beyond the scope of this paper.

Throughout this paper we shall refer to a simple example of the above class of generalization problems, in order to illustrate several approaches to learn-

ing. In this problem, the instances are unordered pairs of simple objects characterized by three properties. Each object is described by its shape (e.g., square, circle, triangle), its color (e.g., red, orange, yellow), and its size (e.g., large, small). The instance language will describe each instance as an unordered pair of feature vectors, each of which specifies the size, color, and shape of an object. For example, Instance_1 below describes an instance in this language.

Instance_1: {(Large Red Square) (Small Yellow Circle)}.

Generalizations of these instances will be represented in a similar fashion, except that we may indicate that the color, size, or shape of an object is unimportant by replacing the value of that feature by a question mark. Thus, the following generalization represents the set of all instances containing one small circle and one large object.

Generalization_1: {(Small ? Circle) (Large ? ?)}

We define the matching predicate for this instance language and generalization language so that a generalization matches an instance provided the features specified in the generalization have counterparts in the features specified in the instance. Thus, Generalization_1 matches Instance_1 (note the instances and generalizations are *unordered* pairs). More precisely, in this example problem we will say that a generalization, g, matches an instance, i, if and only if there is a mapping from the pair of feature vectors of g onto the pair of feature vectors of i, such that the restrictions on feature values given in g are consistent with the feature values of i. Here a feature restriction in g is consistent with a feature value in i if either (a) the feature restriction in g is identical to the feature value in i, or (b) the feature restriction in g is a question mark.

3. Generalization as Search

The above generalization problem is essentially a search problem. The generalization language corresponds to an hypothesis space (search space) of possible solutions, and the learning task is to examine this hypothesis space, subject to constraints imposed by the training instances, to determine plausible generalizations. This characterization of generalization as search is used below to describe generalization *methods*, independent of the particular generalization and instance languages used. This characterization leads to a useful classification and comparison of various systems.

3.1. The partial ordering

A key characteristic of the above generalization problem is that there is an important structure inherent to the generalization language for every such problem. This structure, which has been described previously for

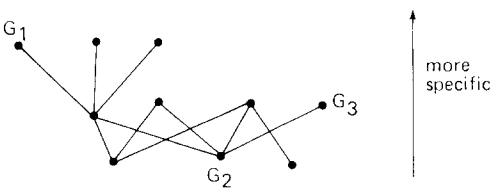
individual generalization languages [5, 8, 11, 15, 18] is based on the relation 'more-specific-than', defined as follows.

More-specific-than relation
Given two generalizations, G_1 and G_2, G_1 is *more-specific-than* G_2 if and only if $\{i \in I \mid M(G_1, i)\} \subseteq \{i \in I \mid M(G_2, i)\}$, where I is the set of all instances describable in the instance language, and M is the matching predicate.

In other words, G_1 is more-specific-than G_2 if and only if G_1 matches a proper subset of the instances that G_2 matches. This relation partially orders the hypothesis space through which the learning program must search. Notice the above definition of this relation is extensional—based upon the instance sets that the generalizations represent. In order for the more-specific-than relation to be practically computable by a computer program, it must be possible to determine whether G_1 is more-specific-than G_2 by examining the descriptions of G_1 and G_2, without computing the (possibly infinite) sets of instances that they match. This requirement places restrictions upon the nature of generalization languages for which some of the methods below are suited.

A portion of the partially ordered generalization language for the example problem is shown in Fig. 1. Here G_1 is more-specific-than G_2: the constraints in G_2 are logically implied by those in G_1, and therefore any instance which matches G_1 must also match G_2. In contrast, G_3 and G_1 are not comparable generalizations according to the more-specific-than relation: although the sets of instances characterized by G_3 and G_1 intersect, neither set contains the other.

The more-specific-than relation defined above imposes a partial ordering over the generalizations in the hypothesis space. *This partial ordering is important because it provides a powerful basis for organizing the search through the hypothesis space.* Note that the definition of this relation (and the corresponding partial ordering) is dependent only on the defined instance language, generalization language, and matching predicate. It is independent of the particular generalization to be learned and the particular training instances presented.



G_1: {(large Red Circle) (Large ? ?)}
G_2: {(? ? Circle) (Large ? ?)}
G_3: {(? ? Circle) (Large Blue ?)}

FIG. 1. Portion of a partially ordered generalization language.

4. Three Data-Driven Generalization Strategies

If generalization is viewed as a search problem, then generalization methods can be characterized in terms of the search strategies that they employ. Many generalization programs employ search strategies that are *data-driven*, in the sense that they consider discrepancies between the current hypothesis and available data in order to determine appropriate revisions to the current hypothesis. Although no two of these programs employ exactly the same strategy, it is informative to group them into classes whose members employ similar strategies and therefore possess similar performance characteristics. The aim of this section is not to compare alternative generalization learning *programs*, but rather alternative *classes* of data-driven strategies that existing programs implement in various ways, for various generalization languages. We consider three such classes of search strategies here. A prototypical program is described for each class, and the characteristics of the prototype examined. The capabilities and efficiency of the classes are then compared in terms of these prototypes.

One data-driven strategy for generalizing from examples is depth-first search through the hypothesis space. Programs that can be characterized in this way include [21] and the RULEMOD portion of the Meta-DENDRAL program as described in [4]. In this strategy a single generalization is chosen as the *current best hypothesis* for describing the identity of the target generalization. This current hypothesis is then tested against each newly presented training instance, and is altered as needed so that the resulting generalization is consistent with each new instance. Each such alteration yields a new current hypothesis, and corresponds to one step in a data-driven, depth-first search through the hypothesis space.

A prototypical depth-first search strategy can be described as shown in Fig. 2.

Fig. 3 illustrates the depth-first search strategy in the context of the example problem described earlier. This figure shows the effect of two positive training instances. Here the first positive training instance leads to initializing the current best hypothesis to CBH_1, which matches no instances other than the first positive instance. When the second positive instance is observed, CBH_1 must be revised so that it will match the new positive instance. Notice that there are many plausible revisions to CBH_1 in addition to CBH_2, shown in the figure. Systems such as [4] and [21] use domain-specific heuristics to determine which of the possible revisions to select when many are plausible.

Fig. 4 illustrates the effect of a third training instance which conflicts with CBH_2. In this case, although CBH_2 could be specialized to exclude the new negative instance, no such revision is consistent with the observed positive instances. Therefore, the system must backtrack to an earlier version of the CBH, reconsidering its previous revisions to determine a revision that will be consistent with the new negative instance as well as the observed positive

Initialize the current best hypothesis, CBH, to some generalization that is consistent with the first observed positive training instance.

for each subsequent instance, *i*
begin
if *i* is a negative instance, and *i* matches CBH
then begin
-Consider ways of making CBH more specific so that *i* no longer matches it.
-Test these possible revisions to find those that match all earlier positive instances.
-Choose one acceptable revision as the new CBH.
end
else if *i* is a positive instance, and *i* does not match CBH,
then begin
-Consider ways of making CBH more general so that *i* matches it.
-Test these possible revisions to find those that do not match any earlier negative instance.
-Choose one acceptable revision as the new CBH.
end
if none of the considered revisions to CBH result in a generalization consistent with previous instances as well as *i*,
then Backtrack to an earlier version of CBH, and try a different branch in the search, and reprocess instances that have been processed since that point.
end

FIG. 2. Depth-first search strategy.

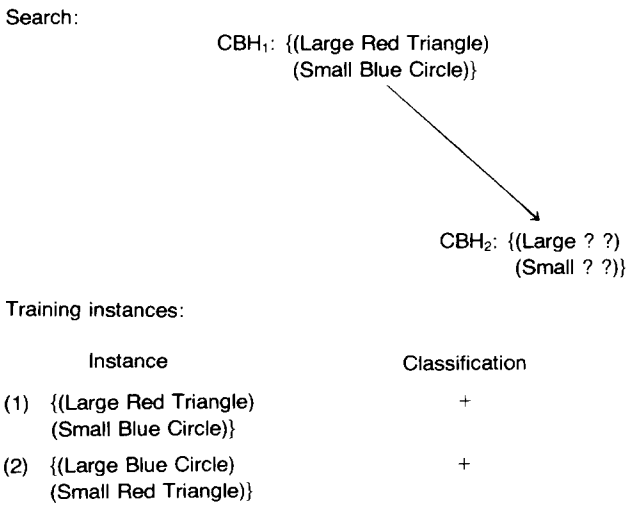


FIG. 3. Depth-first search example.

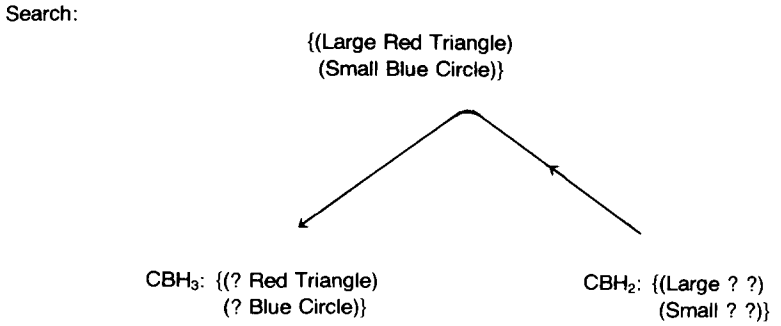


FIG. 4. Depth-first search example continued.

instances. This backtracking step is shown schematically in Fig. 4, and results in this case in the new current hypothesis CBH_3.

There are two awkward characteristics of this depth-first search strategy:

(1) *Cost of maintaining consistency with past instances*
It is costly to test each alteration to the current hypothesis for consistency with past training instances. Some systems (e.g., [21]) sacrifice assured consistency with past instances by not reexamining them when the current hypothesis is altered. Others (e.g., [4]) test past instances, and therefore require progressively longer computations for each successive training instance.

(2) *Need to backtrack*
Once the program has determined a set of acceptable alterations to the current generalization, it must choose one of these as the new current hypothesis. In the event that subsequent instances reveal an incorrect choice has been made, the program must backtrack to reconsider previously processed training instances and generalizations.

4.2. Specific-to-general breadth-first search

In contrast to depth-first search programs, programs which employ a breadth-first strategy maintain a set of *several alternative hypotheses.* Systems which fall into this class include those reported in [6, 15, 18]. Each of the programs takes advantage of the general-to-specific partial ordering to efficiently organize the breadth-first search. Starting with the most specific generalizations, the search

is organized to follow the branches of the partial ordering so that progressively more general generalizations are considered each time the current set must be modified. The set of alternative plausible hypotheses computed by this specific-to-general breadth-first search is the set (which we shall call S) of maximally specific generalizations consistent with the observed training instances; that is

$S = \{s \mid s$ is a generalization that is consistent with the observed instances, and there is no generalization which is both more specific than s and consistent with the observed instances$\}$.

A prototypical specific-to-general breadth-first search is described in Fig. 5.

Notice that this algorithm involves comparing generalizations in order to determine whether one is more general than another. The generalization language must allow making this test efficiently; that is, the test should be made by examining the descriptions of the two generalizations directly, without having to consider explicitly the sets of instances that they represent. This requirement represents a restriction on the kind of generalization languages for which this approach is practical.

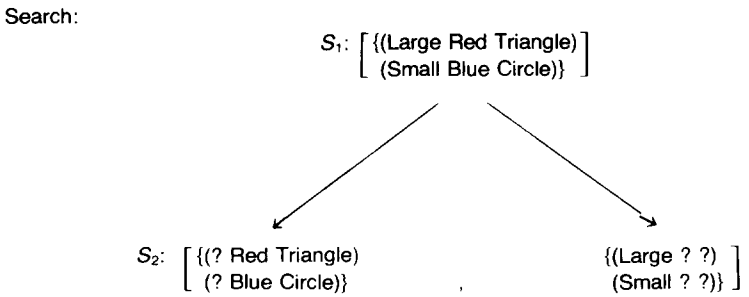
Fig. 6 illustrates this search strategy, using the same two positive instances considered in Fig. 3. The set S_1 is determined in response to the first positive instance. It contains the most specific generalization consistent with the observed instance. S_1 is then revised in response to the second positive instance as shown in Fig. 6. Here, the generalization in S_1 is generalized along each branch of the partial ordering, to the extent needed to match the new positive instance. The resulting set, S_2, is the set of maximally specific describable generalizations consistent with the two observed positive instances.

Fig. 7 illustrates the effect of a subsequent negative training instance. In this case, one of the members of S_2 was found to match the negative instance, and was therefore removed from the revised set of current hypotheses, S_3. Notice that

```
Initialize the set of current hypotheses, S, to the set of maximally specific generalizations
that are consistent with the first observed positive training instance.

for each subsequent instance, i
  begin
    if i is a negative instance,
      then Retain in S only those generalizations which do not match i.
   else if i is a positive instance,
      then begin
           -Generalize members of S that do not match i along each branch of the partial
            ordering but only to the extent required to allow them to match i.
           -Remove from S any element that either (1) is more general than some other
            element in S or (2) matches a previously observed negative instance.
           end
  end
```

FIG. 5. Breadth-first search strategy.



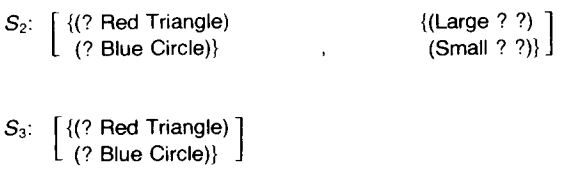
Training instances:

	Instance	Classification
(1)	{(Large Red Triangle) (Small Blue Circle)}	+
(2)	{(Large Blue Circle) (Small Red Triangle)}	+

FIG. 6. Specific-to-general breadth-first search example.

there is no possibility of finding an acceptable specialization of the offending generalization, since by definition, no more specific generalization is consistent with the observed positive instances. At the same time, no further generalization is acceptable since this will also match the new negative instance.

In general, positive training instances force the set S to contain progressively more general generalizations. Each revision to (further generalization of) a member of S corresponds to searching deeper into the partial ordering along one branch of the breadth-first search. Negative instances eliminate general-



Training instances:

	Instance	Classification
(3)	{(Large Blue Triangle) (Small Blue Triangle)}	−

FIG. 7. Breadth-first search example continued.

izations from S, and thereby prune branches of the search which have become overly general. This search proceeds *monotonically* from specific to general generalizations.

One advantage of this strategy over depth-first search stems from the fact that the set S represents a threshold in the hypothesis space. Generalizations more specific than this threshold are not consistent with all the observed positive instances, whereas those more general than this threshold are. Thus, when a generalization in S must be revised, it can only be made more general, and this revision therefore need not be tested for consistency with past *positive* instances. Revisions must still, however, be tested against previous negative instances to assure that the revised generalization is not overly general.

4.3. Version Space strategy

The version space strategy for examining the hypothesis space involves representing and revising the set of *all hypotheses* that are describable within the given generalization language and that are consistent with the observed training instances. This set of generalizations is referred to as the *version space* of the target generalization with respect to the given generalization language and observed training instances. The term version space is used to refer to this set because it contains all plausible versions of the emerging concept.

This strategy begins by representing the set of all generalizations consistent with the first positive training instance, then eliminates from consideration any generalization found inconsistent with subsequent instances. Programs that implement this strategy for various generalization languages are described in [10–12].

The version space approach is feasible because the general-to-specific ordering of generalizations allows a compact representation for version spaces. In particular, a version space can be represented[1] by two sets of generalizations: the set S as defined above, and the dual set G, where

$G = \{g \mid g$ is consistent with the observed instances, and there is no generalization which is both more general than g, and consistent with the instance$\}$.

Together, the sets S and G precisely delimit the version space.[2] It is thus possible to determine whether a given generalization is contained in the version space delimited by sets S and G:

A generalization, x, is contained in the version space represented by S

[1] The version space is 'represented' in the sense that it is possible to generate and recognize any generalization in the version space by examining its representation.

[2] The version space relative to any given set of training instances forms a convex set with respect to the partial ordering of the search space. For a formal description and analysis of this approach, see [11].

and G if and only if

(1) x is more specific than or equal to some member of G, and

(2) x is more general than or equal to some member of S.

The set S is computed in a manner similar to that described for the specific-to-general breadth-first search strategy described above. The set G can be computed by conducting a second, complementary, breadth-first search from general to specific generalizations. The version space strategy can thus be viewed as an extension of the above breadth-first search strategy into a bi-directional search, and can be described as shown in Fig. 8.

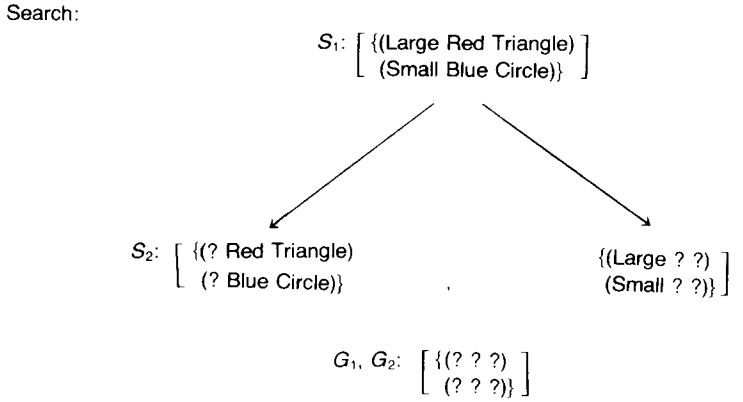
Fig. 9 shows the effect of the same two positive instances shown in the previous examples. The situation is very similar to that for the breadth-first search, except that the additional set G is initialized as shown. The generalization used to initialize the set G is the most general generalization describable within the given language, and matches every possible instance. Because it is consistent with the two positive training instances shown in this figure, the set G is unaltered by these instances.

Fig. 10 illustrates the effect of a negative training instance on the version space. Here the set S_2 is revised as in the breadth-first search. The set G_2 is also revised since the negative instance reveals that the current member of G_2 is overly general. The generalization in G_2 is therefore specialized along all possible branches of the partial ordering that lead toward some member of S_3.

Initialize the sets S and G, respectively, to the sets of maximally specific and maximally general generalizations that are consistent with the first observed positive training instance.

for each subsequent instance, *i*
  begin
    if *i* is a negative instance,
      then begin
        —Retain in S only those generalizations which do not match *i*.
        —Make generalizations in G that match *i* more specific, only to the extent required so that they no longer match *i*, and only in such ways that each remains more general than some generalization in S.
        —Remove from G any element that is more specific than some other element in G.
        end
  else if *i* is a positive instance,
      then begin
        —Retain in G only those generalizations that match *i*.
        —Generalize members of S that do not match *i*, only to the extent required to allow them to match *i*, and only in such ways that each remains more specific that some generalization in G.
        —Remove from S any element that is more general than some other element in S.
        end
  end

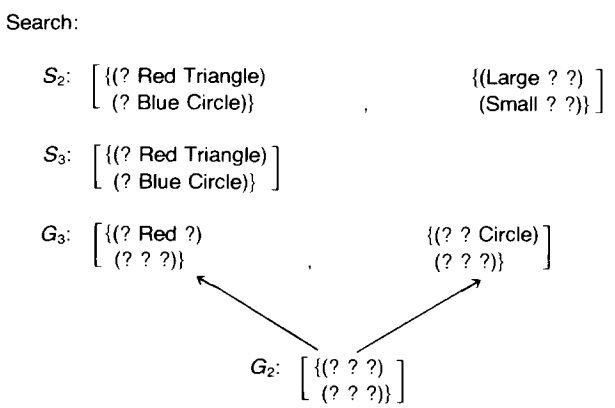
FIG. 8. Version space strategy.



Training instances:

	Instance	Classification
(1)	{(Large Red Triangle) (Small Blue Circle)}	+
(2)	{(Large Blue Circle) (Small Red Triangle)}	+

FIG. 9. Version space example.



Training instances:

	Instance	Classification
(3)	{(Large Blue Triangle) (Small Blue Triangle)}	–

FIG. 10. Version space example (continued).

Along each such branch, it is specialized only the extent required so that the generalization no longer matches the new negative instance.

The revised S and G sets illustrated in Fig. 10 represent the version space of all generalizations in the provided language which are consistent with the three observed training instances. The version space at this point contains the members of S_3 and G_3, as well as all generalizations that lie between these two sets in the partially ordered hypothesis space. Subsequent positive training instances may force S to become more general, while subsequent negative training instances may force G to become more specific. Given enough additional training instances, S and G may eventually converge to sets containing the same description. At this point the system will have converged to the only consistent generalization within the given generalization language.

As with specific-to-general breadth first search, the version space approach is practical only for problems in which the 'more-specific-than' relation can be computed by direct examination of described generalizations. There is also a minor theoretical restriction on the form of the generalization language: in order for the sets S and G to correctly delimit any version space that can arise, every chain in the partially ordered generalization language must have a most specific and most general member.

The advantage of the version space strategy lies in the fact that the set G summarizes the information implicit in the negative instances that bounds the acceptable level of generality of hypotheses, while the set S summarizes the information from the positive instances that limits the acceptable level of specialization of hypothesis. Therefore, testing whether a given generalization is consistent with all the observed instances *is logically equivalent* to testing whether it lies between the sets S and G in the partial ordering of generalizations.

The version space method is assured to find all generalizations (within the given generalization language) that are consistent with the observed training instances, independent of the order of presentation of training instances. The sets S and G represent the version space in an efficient manner, summarizing the information from the observed training instances so that no training instances need be stored for later reconsideration.

4.4. Capabilities

In comparing alternative strategies for generalization, the important issues concern relative capabilities rather than efficiency. The major differences in capabilities among the above three data-driven strategies derive from the number of plausible generalizations carried along at each step, and from the use of the partial ordering in guiding the search. We consider two desirable capabilities for a generalization program:

(1) The ability to detect the point at which the target generalization is

completely determined by the training instances, and, when necessary, to use incompletely determined generalizations in a reasonable manner.

(2) The ability to direct the presentation of training instances to obtain informative instances.

4.4.1. *Using incompletely learned generalizations*

One important capability for learning programs is the ability to detect when the observed training data are sufficient to precisely determine the target generalization. That is, to detect the point at which only a single generalization from the provided language remains consistent with the observed data. Of course the generalization is 'learned' at this point only under the assumption that the generalization language contains a correct description of the generalization, and that the training instances are correct. The capability to detect this condition is important if the learned information is to be later applied to classify unknown instances. Equally important is the capability to make use of incompletely learned generalizations when only limited training data are available.

The version space strategy provides an easy method for detecting the point at which a generalization is completely determined by a set of training instances, with respect to the given generalization language. This condition is satisfied if and only if the computed sets S and G are equal and contain only one generalization. In contrast, it is difficult to recognize this condition when maintaining only a single current hypothesis, as with the depth-first search strategy, or when maintaining only the set S, as with the breadth-first search strategy.

Because availability of training instances is limited in many domains, and because for some generalization languages no finite set of training instances is sufficient to determine a unique generalization[3], it is crucial to be able to apply incompletely learned generalizations in a reasonable way. For example, suppose that the training instances shown in the previous figures are the only training instances available for that problem. Consider the task of using what has been learned thus far in order to classify the three new instances shown in Fig. 11 as positive or negative.

The sets S and G that represent the version space provide a handle on the problem of representing and using incompletely learned generalizations. Even though the exact identity of the target generalization is not fully determined by the three training instances in the preceding example, it *is* assumed that the correct description of the target generalization lies somewhere within the version space delimited by S_3 and G_3 of Fig. 10. Therefore, if a new instance matches every generalization in the version space (equivalently, if it matches every element in the set S), then it can be classified as a positive instance with

[3] Finite sets of training instances from an infinite instance language are not in general sufficient to determine a unique generalization.

$Instance_1$: {(Small Red Triangle) (Large Blue Circle)}
$Instance_2$: {(Large Blue Triangle) (Small Blue Square)}
$Instance_3$: {(Small Red Circle) (Small Blue Circle)}

FIG. 11. Instances with unknown classification.

the same certainty as if a unique generalization had been determined by the training instances. This is the case for $Instance_1$ in Fig. 11.

Similarly, if the instance matches no generalization in the version space (i.e., it matches no element of the set G), then it is certain that the instance does not match any description of the target generalization that would be determined by examining additional instances. This is the case for $Instance_2$ in Fig. 11. Thus, for such instances it is possible to obtain classifications that are just as unambiguous as if the learned generalization had been completely determined by the training instances.

In contrast, instances that match some but not all generalizations in the version space cannot be unambiguously classified until further training instances are available. This is the case for $Instance_3$ in Fig. 11. Of course, by considering outside knowledge or by examining the proportion of generalizations in the version space which match the instance, one might still estimate the classification of such instances.

When an instance is unambiguously classified by the version space, then regardless of which member of the version space is the correct description of the target generalization, the classification of the given instance will be the same. All the observed training instances will therefore receive an unambiguous classification by the associated version space. Surprisingly, even instances which have not been observed during training may receive an unambigrous classification, as does $Instance_2$ in Fig. 11. If the instance has not been observed as a training instance by the learning system, then how can the system produce an unambiguous classification of this instance? Are such unambiguous classifications reliable?

It can be proven that any such unambiguous classification is a correct classification, provided that (1) the observed training instances were correct, and (2) the generalization language allows describing the target generalization. Notice that the generalization language used in our example is biased, in the sense that it does not allow describing every possible set of instances. This biased generalization language, together with the observed data leads to an unambiguous classification of $Instance_2$. Provided that this biased generalization language allows describing the correct generalization, the unambiguous classification of $Instance_2$ is the correct classification. This example provides an interesting insight into the significance of initial biases for allowing inductive leaps during generalization. [13] contains a discussion of the importance of and sources of biases for learning and generalization.

Because the specific-to-general breadth-first strategy computes the set S, this strategy allows unambiguously classifying the same positive instances as the version space strategy. Since it does not compute the set G, however, it cannot distinguish between instances which the version space strategy would classify as negative instances, and those which cannot be unambiguously classified. The breadth-first strategy would therefore be able to classify Instance_1 from Fig. 11 as a positive instance, but would not allow a reliable classification of either Instance_2 or Instance_3.

4.4.2. *Selecting new training instances*

A further capability afforded by computing the sets S and G is the selection of informative new training instances. Consider the following problem: after processing some sequence of training instances, a program is provided a set of further instances, without their classifications as positive or negative instances, and is allowed to request the correct classification of any one of them.

The instance whose classification should be requested in this case (the instance which will provide on the average the most useful information) is the instance which comes closest to matching one half of the generalizations in the version space. Regardless of its classification, finding out its classification will allow rejecting one half of the currently plausible generalizations. Thus, by testing each instance to determine what proportion of the generalizations in the version space it matches, the most informative training instance can be selected.

If instead of selecting from a list of possible instances, the program is able to itself generate at each step an instance that matches half the generalizations in the current version space, then the program can itself generate an optimal[4] sequence of training instances for learning the target generalization.

As a simple illustration of using the represented version space to direct the presentation of training instances, suppose that after being shown the three training instances in the example problem above, the learning program is allowed to request the classification of any one of the instances shown in Fig. 11. In this case, Instance_3 is an instance whose classification would be useful to know—it is an instance that matches some, but not all the members of the current version space. On the other hand, since the classifications of Instance_1 and Instance_2 are already determined by the version space, no new information would be obtained by requesting their classification. Thus, the instances whose classification would be informative are precisely those that cannot be reliably classified by the current version space.

[4] This strategy determines the identity of the target generalization in the shortest possible number of training instances, assuming no prior knowledge of the identity of the target generalization. Choosing instances under this handicap is a much different problem than the problem faced by a teacher who knows the generalization, and must choose good instances. Results from information theory involving optimal binary codes apply here.

The breadth-first strategy also provides some information for selecting new training instances. The strategy of selecting instances which match half the generalizations in the computed set S is reasonable, although less complete than the strategy which takes into account the entire version space.

4.5. Complexity and efficiency

The overall space and time efficiency of each approach is determined by a number of factors, including the order of presentation of training instances, the chosen generalization language and the branching of the associated partial ordering, the cost of matching generalizations to training instances, and the amount of space needed to store generalizations and observed instances.

A complete analysis is beyond the scope of this paper, but it is possible to characterize the time and space complexity as a function of the number of training instances, under reasonable assumptions. In particular, we assume that positive and negative instances are distributed uniformly throughout the sequence of training instances.

Under this assumption, bounds on the time and space complexity of the prototype data-driven strategies described earlier are summarized in Table 1. Here p indicates the number of positive training instances, n indicates the number of negative training instances, s indicates the largest size obtained by the set S, and g represents the largest size obtained by the set G. The time complexity bounds indicate bounds on the number of comparisons between generalizations and instances, and comparisons between generalizations. Notice that for some generalization and instance languages, each such comparison may itself be an NP problem. For example, some structural description languages (e.g., that used in Meta-DENDRAL) involve testing subgraph isomorphism (an NP-complete problem) as part of this comparison.

The complexity of the depth-first strategy stems from the need to reexamine past instances after each revision to the current generalization. Note from the earlier description of this strategy that each time a positive instance forces a change to the current hypothesis, all past negative instances must be examined. Thus, time requirements are $O(n)$ for each such positive instance, or $O(pn)$ in total. Revising the current hypothesis in response to negative instances yields a

TABLE 1. Bounds on processing time and maximum storage costs

Strategy	Processing time	Storage space
Depth-first search	$O(pn)$	$O(p+n)$
Specific-to-general Breadth-first search	$O(spn+s^2p)$	$O(s+n)$
Version space strategy	$O(sg(p+n)+s^2p+g^2n)$	$O(s+g)$

similar result. Because all instances must be stored for later reexamination, the space requirements are linear with the number of observed instances, $O(p + n)$.

For the prototype specific-to-general breadth-first strategy, only negative instances need be stored for later examination, so that space requirements are $O(s + n)$. In the time complexity, the term $O(spn)$ arises because each time that a positive instance alters the set S, each altered hypothesis must be compared against all past negative instances. The term $O(s^2p)$ arises because each revised element of S must be tested to determine whether it is more general than another element of S.

Since the version space strategy computes both S and G, no training instances need be saved, and space complexity is $O(s + g)$. Notice that for this strategy, processing time grows linearly with the number of training instances $(p + n)$, whereas for the other two strategies time grows as the product pn. However, in this case processing time grows as the square of both S and G.

In interpreting the above results it is important to know how the sizes of the sets S and G vary over the training sequence. For the generalization languages for which the version space strategy has been implemented, these sets have been observed to first grow in size, then level off, and finally decrease in size as the version space converges toward the correct description of the target generalization. Under such conditions, the dominant term in determining time complexity is the first term in each of the expressions in Table 1. The exact sizes of the sets S and G depend, of course, upon the nature of the generalization language.

A further consideration in determining overall efficiency which has not been considered here is the effect of ordering and selection of training instances. Short, informative training sequences certainly lower demand for computer resources, as well as demands on the supplier of these instances. By investing some time in ordering or selecting training instances, it is possible that a program might lower its total resource requirements. A related issue is the (not well understood) possibility of controlling the sizes of the sets S and G by prudent ordering and selection of training instances.

5. Other Generalization Strategies

The generalization strategies surveyed in the previous section are *data-driven* in the sense that revisions to current hypotheses are made in response to—and directed by—observed discrepancies with the data. This section notes two other classes of generalization strategies that have been used successfully in various domains.

5.1. Generate-and-test strategies

Data-driven search involves considering discrepancies between the current hypotheses and available data, in order to determine appropriate revisions to

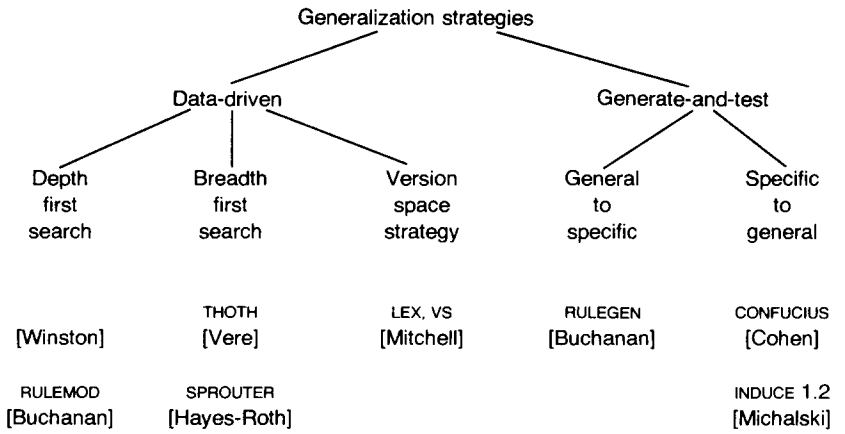


FIG. 12. Some classes of search strategies for generalization.

the current hypotheses. An alternative class of search strategies, which we shall call *generate-and-test* strategies, generates new hypotheses according to a predetermined procedure that is *independent* of the input data.[5] Each newly generated hypothesis is then tested against the entire set of available training data, and identified as either an acceptable generalization, a node to be expanded further by the generator, or a node to be pruned from the search.

Generate-and-test strategies typically consider all available training instances at each step of the search to test newly generated hypotheses. Because they judge the generated hypotheses by their performance over many instances, rather than making decisions based upon individual training instances, they can accommodate quite severe errors in the training data. On the other hand, generate-and-test strategies are not well suited to incremental processing of training data—should unexpected data become available, the generate-and-test search may have to be completely reexecuted. Furthermore, since the generation of hypotheses is not influenced by the data, the search can be quite branchy and expensive.

An interesting combination of generate-and-test and data-driven search procedures is found in the Meta-DENDRAL program [4]. One portion of the program, called RULEGEN [4], conducts a coarse generate-and-test search to form approximate rules of mass spectroscopy based upon highly unreliable training instances. These approximate rules are then used as starting points for a data-driven strategy (either RULEMOD [4] or VS [11]) which conducts a more

[5] This distinction between generate-and-test and data-driven methods is similar to the distinction in [17] between "the rule induction version of the generate and test method" and the "rule induction version of the heuristic search method".

detailed search to refine each rule, using both the original training data and additional available data. Thus, the advantages of generate-and-test search for dealing with inconsistent data are blended with the advantages of data-driven search for a more focused search based on incremental use of the data.

Some generate-and-test strategies for generalization follow the partial ordering of the hypothesis space to control hypothesis generation. [1] describes and compares two such generalization strategies—one that searches from general to specific hypotheses, and one that searches from specific to general. Fig. 12 shows the relationship among the search strategies employed by several existing generalization programs.

5.2. Statistical pattern recognition

The field of statistical pattern recognition deals with one important subclass of generalization problems. In this subclass, the instances are represented by points in n-space, and the generalizations are represented by decision surfaces in n-space (e.g., hyperplanes, polynomials of specified degree). The matching predicate corresponds to determining whether a given point (instance) lies on one side or another of a given decision surface (generalization). The field of Statistical Pattern Recognition has developed very good generalization methods for particular classes of decision surfaces. Many of these methods are relatively insensitive to errors in the data and some have well understood statistical convergence properties, under certain assumptions about the probability distribution of input instances.

In contrast to work in Statistical Pattern Recognition, work on the generalization problem within Artificial Intelligence has focused on problems involving a different class of instance and generalization languages. These languages are incompatible with numerically oriented representations that describe objects as feature vectors in n-space. For example, Winston's program [21] for learning descriptions of simple block structures such as arches and towers, represents instance block structures in terms of their component blocks and relationships among these. In this domain the natural representation for instances is a generalized graph rather than a feature vector. Even the simple generalization problem used as an example in this paper cannot be mapped directly into points and decision surfaces in n-space. Many of the methods of Statistical Pattern Recognition are specialized to numerical feature vector representations, and therefore cannot be applied to these other representations. As a result, methods such as those described in this paper have been developed to handle these new representations.

6. Further Issues

This section notes several issues suggested by the preceding discussion, which relate to significant open problems in machine learning.

6.1. The generalization language

In order to compare approaches that employ different generalization languages we have described strategies and stated results in terms independent of the generalization language used. The choice of a generalization language does, however, have a major influence on the capabilities and efficiency of the learning system.

In choosing a generalization language, the designer fixes the domain of generalizations which the program may describe, and therefore learn. Most current systems employ generalization languages that are biased in the sense that they are capable of representing only some of the possible sets of describable instances. With the choice of a generalization language the system designer builds in his biases concerning useful and irrelevant generalizations in the domain. This bias constitutes both a strength and a weakness for the system: If the bias is inappropriate, it can prevent the system from ever inferring correct generalizations; if the bias is appropriate, it can provide the basis for important inductive leaps beyond information directly available from the training instances. The effect of a biased generalization language on classifying unobserved instances was illustrated in the earlier section on utilizing partially learned generalizations. [13] provides a general discussion of the importance of bias in learning.

The choice of 'good' generalization languages and the impact of this choice on the selection of a good learning strategy is poorly understood at present. Methods by which a program could automatically detect and repair deficiencies in its generalization language would represent a significant advance in this field.

In addition to influencing system capabilities, the choice of generalization language also has a strong influence on the resource requirements of the system. For example, the complexity of the matching predicate for generalizations represented by graphs can be exponential, while the complexity for generalizations represented by feature vectors is linear. Secondly, a language for which the general-to-specific ordering is shallow and branchy will typically yield larger sets S and G than a language in which the ordering is narrow but deep. In particular, the introduction of disjunction into the generalization language greatly increases the branching in the partial ordering, thereby aggravating the combinatorial explosion faced by the learning program.

6.2. Using expectations and prior knowledge

In this discussion we defined 'acceptable' generalizations primarily in terms of consistency with the training data. Generalizations may also be judged in terms of consistency with prior knowledge or expectations. As noted above, one method of imposing such expectation-based, or model-based constraints on a learning system is to build them into the generalization language. A second method is to build them into the generator of hypotheses, as is done in some

generate-and-test searches. In most existing programs the blending of expectations together with constraints imposed by the training data is either done in an ad hoc manner or not done at all. In complex systems the constraints imposed by prior knowledge may be critical to making appropriate inductive leaps, and to controlling the combinatorics inherent in learning. Developing general methods for combining prior knowledge effectively with training data to constrain learning is a significant open problem.

6.3. Inconsistency

In order to simplify the analysis attempted above, it has been necessary to consider only problems in which the generalization language contains some generalization consistent with every training instance. This condition might not be satisfied if either (1) the generalization language is insufficient to describe the target generalization, or (2) the training instances contain errors. In general, there will be no way for the program to determine which of these two problems is the cause of the inconsistency. In such cases, the learning program must be able to detect inconsistency, and recover from it in a reasonable way.

Statistical methods typically deal with inconsistency better than the descriptive methods considered here. As noted earlier, generate-and-test search procedures appear better suited to deal with inconsistency since they base the selection among alternative hypotheses on sets of training instances rather than single instances. Some data-driven strategies have been extended to deal with inconsistent data [5, 11]. Inconsistency is unavoidable in many real-world applications. Well-understood methods for learning in the presence of such inconsistency are needed.

6.4. Partially learned generalizations

As a practical matter, it is essential to develop methods for representing and reasoning about 'partially' learned generalizations. It is unlikely in realistic applications that sufficient training data will be available to fully determine every needed generalization. Therefore, the problem of representing and utilizing incompletely learned generalizations is critical to using generalization methods for practical applications. The techniques noted above for dealing with this issue constitute an initial approach to the problem. Extending these ideas to take advantage of prior knowledge of the domain, and to operate in the presence of inconsistency are important open problems.

7. Summary

The problem of generalization may be viewed as a search problem involving a large hypothesis space of possible generalizations. The process of generalization can be viewed as examining this space under constraints imposed by the

training instances, as well as prior knowledge and expectations. In this light, it is informative to characterize alternative approaches to generalization in terms of the strategy that each employs in examining this hypothesis space.

A general-to-specific partial ordering gives structure to the hypothesis space for generalization problems. Several data-directed generalization strategies have been described and compared in terms of the way in which they organize the search relative to this partial ordering. This examination leads to a comparison of their relative capabilities and computational complexity, as well as to a useful perspective on generalization and significant topics for future work.

ACKNOWLEDGMENT

The ideas presented in this paper have evolved over discussion with many people. John S. Brown, Bruce Buchanan, John Burge, Rick Hayes-Roth, and Nils Nilsson have provided especially useful comments on various drafts of this paper. This work has been supported by NIH under grant RR-643-09, and by NSF under grant MCS80-08889.

REFERENCES

1. Banerji, R.B. and Mitchell, T.M., Description languages and learning algorithms: A paradigm for comparison, *Internat. J. Policy Analysis Informat. Systems* **4** (1980) 197.
2. Brown, J.S., Steps toward automatic theory formation, *Proc. IJCAI* **3** (1973) 20–23.
3. Bruner, J.S., Goodnow, J.J. and Austin, G.A., *A Study of Thinking*. (Wiley, New York, 1956).
4. Buchanan, B.G. and Mitchell, T.M., Model-directed learning of production rules, in: D.A. Waterman and F. Hayes-Roth (Eds.), *Pattern-Directed Inference Systems* (Academic Press, New York, 1978).
5. Hayes-Roth, F., Schematic classification problems and their solution, *Pattern Recognition* **6** (1974) 105–113.
6. Hayes-Roth F., and Mostow, D., An automatically compliable recognition network for structured patterns, *IJCAI* **4** (1975) 356–362.
7. Hunt, E.B., *Artificial Intelligence* (Academic Press, New York, 1975).
8. Michalski, R.S., AQVAL/1—Computer implementation of a variable valued logic system VL1 and examples of its application to pattern recognition, *Proc. 1st Internat. Joint Conf. Pattern Recognition*, Washington, DC (1973) 3–17.
9. Minsky, M. and Papert, S., *Perceptions* (MIT Press, Cambridge, MA, 1969).
10. Mitchell, T.M., Version spaces: A candidate elimination approach to rule learning, *IJCAI* **5** (1977) 305–310.
11. Mitchell, T.M., Version spaces: An approach to concept learning, Ph.D. Thesis, Stanford University, December 1978; also Stanford CS Rept. STAN-CS-78-711, HPP-79-2.
12. Mitchell, T.M., Utgoff, P.E. and Banerji, R.B., Learning problem-solving heuristics by experimentation, in: R. Michalski et al. (Eds.), *Machine Learning* (Tioga Press, Palo Alto, 1982).
13. Mitchell, T.M. The need for biases in learning generalizations, Rutgers Computer Science Tech. Rept. CBM-TR-117.
14. Nilsson, N.J., *Learning Machines* (McGraw-Hill, New York, 1965).
15. Plotkin, G.D., A note on inductive generalization, in: B. Meltzer and D. Michie (Eds.), *Machine Intelligence* **5** (Edinburgh University Press, Edinburgh, 1970) 153–163.
16. Popplestone, R.J., An experiment in automatic induction, in: B. Meltzer and D. Michie (Eds.), *Machine Intelligence* **5** (Edinburgh University Press, 1970) 204–215.

17. Simon H.A., and Lea, G., Problem solving and rule induction: a unified view, in: L.W. Gregg (Ed.), *Knowledge and Cognition* (Erlbaum, Potomac, MD, 1974) 105–127.
18. Vere, S.A., Induction of concepts in the predicate calculus, *IJCAI* **4**, Tbilisi, USSR (1975) 281–287.
19. Vere, S.A., Inductive learning of relational productions, in: D.A. Waterman and F. Hayes-Roth (Eds.), *Pattern-Directed Inference Systems* (Academic Press, New York, 1978).
20. Waterman, D.A., Generalization learning techniques for automating the learning of heuristics, *Artificial Intelligence* **1**(1,2) (1970) 121–170.
21. Winston, P.H., Learning structural descriptions from examples, in P.H. Winston (Ed.), *The Psychology of Computer Vision* (McGraw-Hill, New York, 1975).

Received January 1980; revised version received June 1981