Machine Learning 2: 103-138, 1987
© 1987 Kluwer Academic Publishers, Boston - Manufactured in The Netherlands

Experiments with Incremental Concept Formation: UNIMEM

MICHAEL LEBOWITZ (LEBOWITZ@CS.COLUMBIA.EDU)
Department of Computer Science, Columbia University, New York, NY 10027, U.S.A.

(Received: November 14, 1986)
(Revised: June 9, 1987)

Keywords: Concept formation, learning by observation, generalization, conceptual clustering

Abstract. Learning by observation involves automatic creation of categories that summarize experience. In this paper we present UNIMEM, an artificial intelligence system that learns by observation. UNIMEM is a robust program that can be run on many domains with real-world problem characteristics such as uncertainty, incompleteness, and large numbers of examples. We give an overview of the program that illustrates several key elements, including the automatic creation of non-disjoint concept hierarchies that are evaluated over time. We then describe several experiments that we have carried out with UNIMEM, including tests on different domains (universities, Congressional voting records, and terrorist events) and an examination of the effect of varying UNIMEM's parameters on the resulting concept hierarchies. Finally we discuss future directions for our work with the program.

1. Introduction

Learning from observation is a task that is important in domains where examples are not pre-classified, but where one still wishes to detect general rules and intelligently organize examples. In this paper we discuss UNIMEM, a system that learns from observation by noticing regularities among examples and organizing them into a generalization hierarchy. We view UNIMEM both as implementing an algorithm for concept formation and as a prototype intelligent information system that can incorporate large amounts of data into memory and retrieve appropriate information in response to user queries. UNIMEM is not intended to be a psychological model *per se*, since it deals with a task more data-intensive than people are likely to perform. However, in developing the program we have made use of techniques derived by observing human behavior.

The task of UNIMEM is to take a series of examples (or *instances*) that are expressed as collections of features and build up a generalization hierarchy of concepts. For example, UNIMEM might use information about a collection of universities to inductively determine the concepts of Ivy League universities, European technical universities, and so forth, and determine which examples are described by which concepts. The point of creating such concept descriptions is that they allow a performance element using the output of the program to make inferences about new examples based on partial information.

Successful learning from real-world input must deal with several constraints. The key features that characterize the operation of UNIMEM are:

- It learns *by observation*; it is not explicitly told how examples should be grouped into categories;
- It is *incremental*; output must be available after processing each example; it cannot wait for all the input;
- It must handle *examples in large numbers* (currently hundreds, eventually more);
- Its generalizations are *pragmatic*; they need not perfectly describe all the instances they cover.[1]

Although certain learning systems have dealt with tasks having some of these characteristics, little work has been concerned with all of them. However, all seem to characterize human concept formation and all seem valuable for learning in complex real-world domains. We constantly receive new examples and the world is not perfectly regular.

The task of UNIMEM is basically that of conceptual clustering as presented by Michalski and Stepp (1983) and Fisher and Langley (1985), but our work also draws upon research in learning from examples (e.g., Winston, 1975; Mitchell, 1982; Dietterich & Michalski, 1986). However, in a learning by observation setting, one must consider not just how to compare examples, but also decide which examples to compare. This decision largely determines the concepts that one creates. We make the assumption that similarities among natural occurring examples reflect meaningful regularities in the world, an assumption that we discuss at length elsewhere (Lebowitz, 1986a).

The name UNIMEM is derived from the phrase UNIversal MEMory model, which reflects our goal of generality. We would like the system to be easily applicable to new domains, at least those where a feature-based representation is adequate. Domains in which UNIMEM has been used

[1]Pragmatic generalization is crucial in dealing with uncertain, incomplete, or inconsistent data, where apparently equivalent situations may produce different results.

include: U.S. states, Congressional voting records, software evaluations, biological data, football plays, universities, terrorist events, census data, and financial data. In the following sections we provide an overview of the UNIMEM learning algorithm, along with an example of the system in operation, and then describe several experiments that we have performed with the program. These include both examining the system's behavior in several different domains and a study of the effects of varying UNIMEM's parameters. We conclude with a discussion of several open research issues and the relation of our work to other research in machine learning.

2. The basic UNIMEM algorithm

UNIMEM takes a series of examples in a domain and organizes them into a permanent long-term memory.[2] The key idea behind the system is Generalization-Based Memory (GBM),[3] a hierarchy of concepts for describing classes of objects. GBM is built up by generalizing from specific examples, which involves both searching memory for similar examples and abstracting out similarities. To illustrate the UNIMEM learning algorithm, we will use examples from the domain of university information. For this domain we collected descriptions of 224 universities. Information was taken from standard reference books and by surveying undergraduate students. In studying learning by observation, we feel that it is important to collect as much information as possible and not prejudge whether any particular piece of information is likely to be useful in generalization.

2.1 UNIMEM's representation of instances and concepts

Input to UNIMEM is a series of examples, or *instances*, given to the program one at a time. An instance is described as a set of *features* that are essentially attribute/value pairs.[4] Each university has attributes such as 'percent of students receiving financial aid,' 'average math SAT score,'

[2]UNIMEM runs in UCI LISP on a DECSystem/2060 and in Portable Standard LISP on an HP 9861 workstation and a DEC VAX-750.

[3]GBM is also used by our other prototype intelligent information system, RESEARCHER, which reads, remembers, and generalizes from patent abstracts (Lebowitz, 1983a, 1986b). The instances in RESEARCHER are more complex than those in UNIMEM, but it can handle fewer examples. The idea of GBM was originally developed for IPP, a program that read and learned from news stories about terrorism (Lebowitz 1980, 1983b); see also Section 3.2 of this paper.

[4]UNIMEM actually uses attribute/facet/value triples. This greatly simplifies its use for frame-based representations. For example, in the terrorist event domain we use attributes and facets to distinguish among features of the different role fillers, e.g, the victim's nationality and the actor's nationality. However, for purposes of clarity, in this paper we have collapsed the attribute and facet fields.

Table 1. Three instances of universities.

Attribute	*Value for Columbia*	*Value for Yale*	*Value for Brown*
State	New-York	Connecticut	Rhode-Island
Location	Urban	Small-City	Urban
Control	Private	Private	Private
Male:Female	7:3	55:45	1:1
No-of-Students	< 5,000	< 5,000	< 5,000
Student:Faculty	9:1	5:1	11:1
SAT-Verbal	625	675	625
SAT-Math	650	675	650
Expenses	> \$10,000	> \$10,000	> \$10,000
%-Financial-Aid	60	40	40
No-Applicants	4,000-7,000	10,000-13,000	10,000-13,000
%-Admittance	30	20	20
%-Enrolled	50	60	50
Academics	5 out of 5	5 out of 5	5 out of 5
Social	3 out of 5	3 out of 5	4 out of 5
Quality-of-Life	3 out of 5	4 out of 5	5 out of 5
Acad-Emphasis	Lib-Arts	History	History
Acad-Emphasis		Biology	Biology
Acad-Emphasis		English	Art-Sciences
Acad-Emphasis		Lib-Arts	

and so forth. Some features, such as quality of social life, make use of arbitrary five point scales. While a simple feature representation is clearly inadequate for many tasks, it allows us to get started very easily on new domains. Table 1 shows the input features for Columbia, Yale, and Brown, three typical instances in the university domain.

The goal of UNIMEM is to recognize similar instances and abstract them to form a hierarchy of generalized concept descriptions. Instances are stored in GBM under the generalizations that describe them. The resulting concept hierarchy can, if desired, be used by a performance system, such as a question-answering program. The manner in which generalizations are related is illustrated in Table 2, which shows part of a concept hierarchy formed by UNIMEM from 150 university instances averaging about 20 features apiece.[5] (The complete hierarchy is available upon request from the author.) The table shows how the basic concept of a university is broken down into a number of more specialized versions. The hierarchical nature of the generalizations is indicated by indentation (e.g., GND60 inherits all the properties of GND2). The English concept descriptions have been

[5]UNIMEM does not require that every instance have a value for every attribute, hence the number of features per instance varies. Also, attributes with multiple values are allowed.

Table 2. A portion of UNIMEM's concept hierarchy for a university run.

```
GND0 {'unusual' universities that are not covered by any generalization}
     [DALLAS-BAPTIST-COLLEGE JUILLIARD MICHIGAN-STATE SUNY-BUFFALO
      UNIVERSITY-OF-MISSISSIPPI VASSAR]
  GND2 {high quality of life and academics; engineering emphasis}
       [CHALMERS-UNIVERSITY-OF-TECHNOLOGY ECOLE-POLYTECHNIQUE PENN-STATE
        SAN-JOSE-STATE UNIVERSITY-OF-CALIFORNIA-SAN-DIEGO
        UNIVERSITY-OF-TEXAS]
    GND60 {large state schools with strong social life}
          [UNIVERSITY-OF-COLORADO UNIVERSITY-OF-MASSACHUSETTS-AMHERST]
    . . .
  GND4 {private universities with high academic level and medium social
        life}
       [ ]
    GND9 {expensive, urban schools with strong applicant SAT scores}
         [HARVARD UNIVERSITY-OF-PENNSYLVANIA]
      GND119 {small schools with low admittance rates}
             [COLUMBIA WESLEYAN]
    GND19 {expensive schools with high enrollment yields}
          [MIT SWARTHMORE]
      GND133 {small schools with very high SATs and low admittance rates}
             [PRINCETON YALE]
      . . .
    . . .
  . . .
```

added by hand. At the top level, we see universities under GND0 that are described by no generalized concepts. Shown beneath GND0 are two generalized concepts, GND2 and GND4. The latter of these, which describes private universities, also has several more specific versions.[6]

In the hierarchy of generalizations that describe concepts of increasing specificity, instances and sub-generalizations are stored using efficient indexing methods.[7] The generalizations themselves are sets of features. Table 3 shows several of the generalizations taken from the hierarchy in Table 2. GND4, the first generalization in the table, can be summarized as 'high-quality private universities,' represented by an appropriate set of features. In this hierarchy, no instances were stored directly under GND4, since those from which it was created had all been used to create sub-generalizations.

[6]The more specific versions of a generalization are referred to as its sub-generalizations.

[7]We have experimented with both discrimination networks (Feigenbaum, 1963) and hash tables for indexing. The exact indexing method is not crucial in most domains; there are rarely a large number of instances under a given generalization, since sub-generalizations tend to be formed as the number of instances grows.

Table 3. Selected concept descriptions for the university domain.

Attribute	*Value*	*Feature Frequency*	*Confidence Level*
GND4 [no instances]			
Quality-of-Life	4 out of 5	1	20.00
Academics	4.5 out of 5	2	17.67
Control	Private	3	16.00
Social	3 out of 5	3	20.00
GND9 [Harvard University-of-Pennsylvania]			
Sat-Math	662.5	1	4.13
%-Financial-Aid	60	1	5.00
Location	urban	4	0.00
Student:Faculty	10:1	5	4.40
Expenses	> $10,000	5	11.00
GND19 [MIT Swarthmore]			
Sat-Verbal	637.5	1	2.72
%-Financial-Aid	45.0	1	2.20
%-Enrolled	55.0	2	1.00
No-of-Students	< 5,000	5	5.00

As part of its representation, UNIMEM includes numeric ratings that indicate its confidence in each feature of each generalization. These numbers start at 0 and can increase or decrease during the processing of later examples, as described in Section 2.2.3. The values in the rightmost column of Table 3 are the confidence levels.[8]

The numbers in the third column of the table are feature frequencies that indicate how often each feature appears in other generalizations. This information is used for predictability analysis, a method for determining which features are likely to indicate a generalization's relevance to new examples. While we will not discuss predictability in depth here – it is discussed more fully in Lebowitz (1983b) – the basic idea is that only certain features should be used to index a concept (because they indicate its relevance), and that these features can be identified efficiently using Generalization-Based Memory. Predictability analysis can also be important in determining causal explanations for generalizations (Lebowitz, 1986c).

Table 3 also shows GND9 and GND19, two more specific versions of GND4. The concept GND9 describes expensive, urban schools and GND19 describes schools that are small and have high verbal SAT scores. Each

[8]Naturally, the decimal places should not be taken too seriously. They are the product of the numeric evaluation procedure used.

of these generalizations has instances (universities) stored with it. When future instances are found to be described by these generalizations, they will be compared to the examples stored there.

The use of a hierarchy of generalizations as a method of memory organization allows efficient storage of information since it supports inheritance. In addition, GBM allows the generalizations and instances relevant for learning to be found efficiently in memory using the algorithm described below. This latter property is largely independent of UNIMEM's feature-based knowledge representation, as we have shown with RESEARCHER (Lebowitz, 1983a, 1986b), a system that uses a more complex representational scheme. The use of concept hierarchies with inheritance is by no means new; semantic networks (Quillian, 1968), frame systems (Minsky, 1975), and MOPs (Schank, 1982) are among many formalisms that incorporate this approach. What distinguishes UNIMEM is the dynamic formation of the concept hierarchy and the use of this hierarchy to guide the development of further concepts.

An important part of the UNIMEM methodology is that the more specialized versions of a given concept need not be mutually exclusive. In Table 2, for example, the two concepts 'schools with high quality of life and academics; engineering emphasis' and 'private universities with high academic level and medium social life' are obviously not mutually exclusive; a university could be described by both concepts. An implication of this is that UNIMEM can store an instance in several places in memory. Most clustering techniques require disjoint categories, but this does not seem to be the best way to maximize the inferential power of the concepts created.

Nor must the categories at a given level cover all the instances. Even if a concept allows default inferencing, its negation may not because the instances not in that category may have little in common. For example, universities that are neither in GND2 nor GND4 above may share no features; hence no default inferences could be made based on nonmembership in those classes.

2.2 Adding new instances to memory

The basic process of incorporating a new instance into GBM makes direct use of the memory organization defined above. UNIMEM's incorporation algorithm for a new instance with a list of *input-features* can be broken into two phases:[9]

[9]The UNIMEM incorporation algorithm includes a number of adjustable parameters, noted by a superscript P in the text. By parameterizing all aspects of UNIMEM, we do not give great meaning to any specific numeric value. In Section 4.1 we will discuss

1. Search GBM for the most specific concept node(s) that the instance matches by calling SEARCH(*root-node*, *input features*).
2. Add the new instance to memory by calling UPDATE(*most-specific-node*, *input-features*) for the node(s) found by SEARCH. This involves comparing the new instance to the ones already stored and generalizing if appropriate.

If desired the search phase could be used independently to retrieve instances that match an input description. This could be done for information retrieval and similar applications.

2.2.1 Searching the generalization hierarchy

As UNIMEM processes a new instance, it first finds the most specific generalizations that describe it. GBM can be viewed as a large discrimination net (Feigenbaum, 1963), so UNIMEM starts with its most general node and carries out a controlled depth-first search to find the most specific generalization(s) that legitimately describe the new instance. When the search begins, none of the input features have been matched to a generalization. As UNIMEM searches down the concept hierarchy, features are gradually accounted for by various generalizations. The major steps of the SEARCH algorithm are:

SEARCH(*node*, *unexplained-features*)

1. If the sum of the distances between the features in *unexplained-features* and those of *node* is 'too large',[P] then *node* does not adequately match the instance; return the empty list.
2. Otherwise, for each potentially relevant sub-node *sx* of *node*, call SEARCH(*sx*, [*unexplained-features* − features of *node*]).
3. If for any *sx*, SEARCH returns a list of nodes that describe the new instance, then return the union of those lists.
4. Otherwise, return the singleton list of *node*. (This case occurs only when each sub-node conflicts with the new instance. Since *node* does not conflict with the new instance, it is the most specific acceptable generalization on this search path.)

the possibility of setting the parameters automatically. The Appendix gives a complete listing of UNIMEM's parameters.

During UNIMEM's search process, feature values can do more than match or mismatch - there can be varying degrees of closeness.[10] Instead of values simply matching or not, we allow the quality of feature matches to vary between 0 (total mismatch) and 1 (perfect match).[11] When UNIMEM matches a new instance to a generalization, it considers whether the sum of the distances between the features in the generalization and those in the new instance is small enough[P] to assume that the generalization describes the instance.[12]

If an instance has feature values that conflict with a generalization, which is allowed as long as the total conflict is not too high, then the instance feature values simply override those in the generalization. This contrasts with many learning techniques, which assume that all the features of a generalization must hold for each instance that it describes. In early experiments with UNIMEM, we found that such an all-or-none matching scheme led to the creation of excessive numbers of slightly different generalizations because new instances did not quite fit under old ones. Allowing contradiction does potentially leave UNIMEM open to problems of the sort described by Brachman (1985), such as describing an instance as "an Ivy-League type school except it's not in the East, not private, not expensive ...". However, as long as we keep the allowed-difference parameter small, this does not appear to happen.

2.2.2 Storing a new instance in memory

Once UNIMEM has retrieved the most specific generalization(s) that a new instance matches, it compares the instance against others already stored with the concept(s) to determine whether further generalizations should be made. The system looks for instances that have features in common with the new one. If it finds one that has enough features in common,[P] it creates a new node by generalizing the common features, and it stores the contributing instances with the new concept. If no sufficiently similar instances are found, it stores the new instance under the existing generalization. The attribute/value representation of UNIMEM normally yields a unique generalization of two instances,[13] but multiple generalizations are

[10]We developed categorization algorithms for numeric input that allowed an all-or-none regimen to work reasonably well (Lebowitz, 1985), but we have since modified UNIMEM to take into account the closeness of values as described here.

[11]The system is set up so that a user can easily define different distance measures for various features, if desired. We currently consider numeric data, ordinal data, and simple hierarchical data.

[12]We also add in a penalty for any feature of the generalization simply missing from the instance. This is possible since instance descriptions can be incomplete.

[13]Exceptions would be if there are multi-valued attributes or if the 'averaging' process described below returns multiple possibilities.

sometimes created by matching a new instance with several existing ones. The main steps in the UPDATE algorithm are:

```
UPDATE(node, new-instance)

 1. Define new-features as the features of new-instance that are not part
    of node (or its parent nodes). This information is retained from
    SEARCH.
 2. If none of the instances currently stored under node have enough^P fea-
    tures with values sufficiently close^P to those of new-instance to warrant
    a new generalization, then store new-instance under node.[14]
 3. Otherwise, for each instance with enough features in common with
    new-instance, create a generalization node comprised of the shared
    features and:
     (a) Store the new node in the node's sub-generalization list.
     (b) Store both instances under the new node.
     (c) Remove the old instance from the original node.
```

In deciding which features to include in a generalization, UNIMEM selects all those in the two instances with values that are sufficiently close.^{P} In those cases where features have slightly different values, UNIMEM uses an 'average' feature value in the generalization. For real-valued features this is the arithmetic or geometric mean; for ordinal attributes it is one of the two values; and for hierarchically-ordered attributes it is the lowest common ancestor.

2.2.3 Evaluating generalizations

As seen above, concepts are generalized by UNIMEM on the basis of only two instances. This can cause the creation of an over-specified generalization if the initial instances share spurious features. Generalizations can be under-specified if the instances had unknown values for relevant features (which is possible, since UNIMEM does not require every instance to have values for each feature). Under-specification is not a problem, since the missing features will simply appear in sub-generalizations. However, concepts must be evaluated when they are potentially relevant to future input in order to remove overly-specific features. This is particularly true in domains where there are a large number of features for each instance, since coincidental matches become inevitable. UNIMEM performs evaluation as

[14]'Enough' is defined as a percentage of the maximum number of features of the two instances being compared.

a normal part of the memory search process, since the generalizations to be evaluated are exactly those that are accessed when a new instance is processed. We simply add the following step to the beginning of the SEARCH algorithm:

- Increase confidence in any features of *node* that are also in *unexplained-features*; decrease confidence in those that are contradicted.[15] Delete any features with confidence levels that go low enough.^{P} Make permanent any features with confidence levels that go high enough^P (e.g., stop modifying their confidence levels).

The evaluation operations are applied to all nodes considered during the SEARCH process, even if they do not ultimately match.

This modification to SEARCH does not lead UNIMEM to entirely eliminate a generalization when it fails to fit later input. Instead, it tries to throw away just the 'bad' (overly specific) parts and keep the 'good' parts. Confidence modification occurs by incrementing confidence levels when new values are close^P to the generalization (in terms of the distance measure) and decrementing them when they are not. The amounts of the increments or decrements depend upon the distance between the feature values of the instance and of the generalization. If a confidence level falls below a negative threshold,^{P} then the system eliminates that feature from the generalization, since it has unreliably appeared in instances when the generalization seemed relevant.[16] Above a specified level^P values are 'frozen' and assumed to be permanently correct.

In some cases the feature evaluation process leads to concepts so general that they no longer provide substantial information. There is no advantage to retaining a category with so few features that no inferences can be made when an instance is matched to it. Thus, UNIMEM eliminates an entire generalization when too few of its features^P remain, defined as a percentage of the number of features in the instances that formed the generalization. When it deletes a generalization, UNIMEM also loses access to the instances and sub-generalizations stored there. This loses instances that are not also stored elsewhere, but if we immediately reindexed the instances with the parent node, then the same instances that initially formed the eliminated generalization would do so again. In the domains that we deal with there are enough input examples so that good concepts will eventually be created, despite losing some information. However, for other domains

[15]*Node* is guaranteed to be 'potentially relevant' by the structure of the algorithm.

[16]Other than removing features, UNIMEM does not use the confidence level in the matching process. An interesting extension might be to give added weight to features with high confidence values.

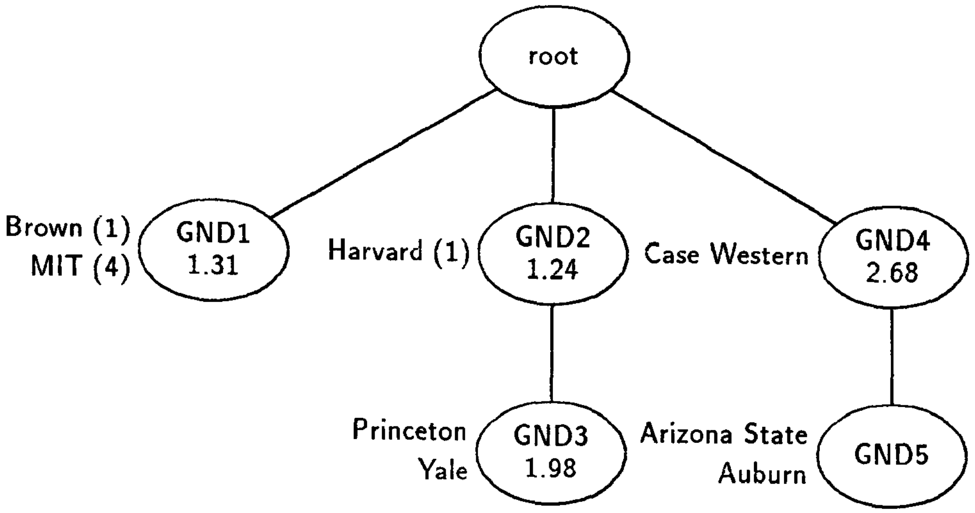

Figure 1. Memory structure at the outset of the sample run. The values shown for GND1, GND2, GND3, and GND4 are the sum of the feature distances between each node and Columbia. The numbers in parentheses are the number of features the instance has in common with Columbia, not including the ones accounted for by generalizations.

different strategies might be appropriate, such as putting deleted instances back into memory after a delay.

2.3 A simple program trace

To illustrate UNIMEM's update algorithm, we will look in detail at how it adds an instance from the university domain to an existing memory.[17] This example will involve the three universities described in Table 1 as well as six others – MIT, Princeton, Harvard, Arizona State, Case Western, and Auburn. Figure 1 shows the structure of UNIMEM's memory after the instances MIT, Brown, Princeton, Harvard, Yale, Arizona State, Case Western, and Auburn have been processed in that order. Table 4 shows the details of the generalizations, including the feature confidences at the beginning and end of the sample run.

We will now describe in some detail how UNIMEM processes a new instance, Columbia. The system begins by searching memory for the most specific generalizations that satisfactorily match the new instance. This

[17]To make pedagogic points, we have set some of UNIMEM's parameters to unrealistic values.

begins by matching Columbia's features with those of GND1. As shown in Figure 1, the total difference between the features of GND1 and Columbia is 1.31. As the parameters were set for this run, the allowed difference is 1.36 (8% of 17 features), so GND1 is considered acceptable. The same is true for GND2 with its 1.24 difference. However, GND2's sub-generalization, GND3, is not acceptable, nor is GND4. Note that since GND4 is not acceptable, its sub-generalization, GND5, is not even considered. The final result of the search is that GND1 and GND2 are the most specific generalizations that match Columbia.

Searching through memory also involves updating confidence levels for features. In this case, if a generalization's feature value is close to that of Columbia, then its confidence level is increased; otherwise it is decreased. The amount of the increment or decrement is based upon the degree of the match or mismatch. Looking at the rightmost two columns for GND1 in Table 4, one can see that the confidence level for the percentage of financial aid goes down, since the generalization value is 45% compared to 60% for Columbia. The remaining confidence levels go up, as the Columbia values are quite close to the values in GND1. The confidence levels for GND5 (not shown in the table) are not adjusted at all, as it is skipped by the search algorithm.

While considering GND2, UNIMEM reduces the confidence level for history as an academic emphasis from -2.0 to -3.0. This causes the confidence level to drop below the threshold for retaining features, so the system deletes the feature from the generalization. In order to maintain correctness, the same feature is added to GND2's sub-generalization, GND3. Also, since the feature was deleted from GND2 during the matching process, this particular feature difference is not considered part of the total discrepancy between GND2 and the new instance, which allows a match with Columbia.

With the search and confidence evaluation phase complete, UNIMEM updates memory by adding Columbia to both GND1 and GND2. In each case, it compares the new instance to those already stored with the generalization to see if there are a significant number of features in common (other than those already accounted for by the generalization). The number of features that Columbia has in common with each relevant instance is shown in parentheses in Figure 1. Columbia shares only one feature with Brown, the first instance under GND1, but it shares four with MIT. Since this is above the parameter for generalizing on this run, UNIMEM creates a new generalization, GND6, which is indexed under GND1. Both MIT and Columbia are stored under the new generalization. Harvard, the only instance under GND2 (the other generalization that Columbia matched), shares only one feature with the new instance, and so no generalization is made. Columbia is simply stored under GND2. Figure 2 shows the

Table 4. Generalizations involved in the sample run.

Attribute	*Value*	*Feature Frequency*	*Initial Confidence*	*Final Confidence*
GND1				
Student:Faculty	5:1	1	5.78	6.75
SAT-Verbal	637.5	1	−2.00	−1.25
%-Financial-Aid	45.0	1	1.60	1.40
%-Admittance	25.0	1	−1.20	−0.60
%-Enrolled	55.0	1	0.80	1.40
Social	3.5 out of 5	1	3.33	4.00
No-of-Students	< 5,000	1	0.00	1.00
Location	Urban	1	−1.00	0.00
Expenses	> $10,000	2	2.00	3.00
Academics	5 out of 5	2	1.33	2.33
Control	Private	2	2.00	3.00
GND2				
%-Financial-Aid	55.0	1	1.20	1.80
%-Admittance	20	1	−2.00	−1.80
Social	3 out of 5	1	2.00	3.00
Quality-of-Life	3.5 out of 5	1	2.00	2.67
Acad-Emphasis	History	1	−2.00	deleted
Acad-Emphasis	Liberal-Arts	1	−2.00	−1.00
Male:Female	65:35	1	−0.68	−0.12
Student:Faculty	7:1	1	3.89	4.88
SAT-Math	675	1	−0.50	0.00
Expenses	> $10,000	2	0.00	1.00
Academics	5 out of 5	2	−0.67	0.33
Control	Private	2	0.00	1.00
GND3, a more specific version of GND2				
SAT-Verbal	662.5	1	0.00	0.25
%-Financial-Aid	45.0	1	0.00	0.00
No-of-Applicants	10,000-13,000	1	0.00	−1.00
%-Enrolled	60	1	0.00	0.20
No-of-Students	< 5,000	1	0.00	1.00
Acad-Emphasis	History	1		0.00
GND4				
Student:Faculty	20:1	1	1.00	1.97
%-Admittance	82.5	1	0.40	−0.60
Academics	3 out of 5	1	0.33	0.00
Acad-Emphasis	Engineering	1	1.00	0.00
GND6, a more specific version of GND1				
Male:Female	75:25	1	–	0.00
%-Financial-Aid	55.0	1	–	0.00
No-of-Applicants	4,000-7,000	1	–	0.00
Quality-of-Life	3 out of 5	1	–	0.00

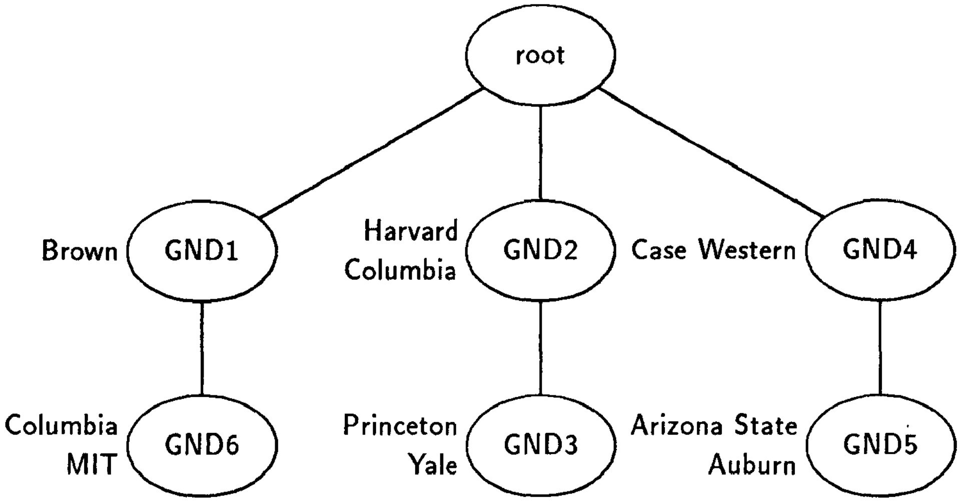

Figure 2. Memory structure after Columbia has been processed.

structure of memory after the processing of Columbia is complete.

Notice that Columbia was not compared against any of the instances stored under GND3, GND4, or GND5. This university may have much in common with some of these instances, but it is much more likely to be similar to those under the matched generalizations. Restricting the set of instances that are matched against is a prime factor in maintaining the efficiency of the algorithm.

This sample run also illustrates the nature of UNIMEM's nondisjoint generalizations. GND1 and GND2 are not mutually exclusive, and the program has matched Columbia with both of them. Essentially, GND1 covers small urban universities with high academic levels and GND2 covers high 'quality of life' liberal arts schools. Columbia can quite logically be considered to exemplify either concept.

2.4 UNIMEM in terms of search and memory organization

Like artificial intelligence programs in general, UNIMEM can be viewed as searching through a space of alternatives. In this case, each state in the space represents an entire concept hierarchy. UNIMEM employs several operators to move through this search space, all of which are driven by the addition of new instances. First, it can simply change the confidence levels of features in concepts that appear relevant to a new instance. Second, it can modify concepts by removing features for which the confidence levels fall too low. Third, it can modify the structure of the generalization hierarchy by adding new concepts when instances are sufficiently similar. Finally, it can delete generalizations (and all their sub-generalizations) when too few features remain after deletions.

Although it is possible to describe UNIMEM in search terms, we feel it is more valuable to describe it in the memory terms that we have been using. The basic data structure of Generalization-Based Memory is the key to its operation. In fact, we feel that more researchers should consider their work in memory terms. Viewing learning from this perspective forces one to consider how the concepts that are created can be efficiently accessed, how memory should be modified, and how the various data structures evolve over time, both in terms of structure and size.

3. Experiments with UNIMEM

An important criterion on which to evaluate any learning system is its generality. In this section we demonstrate UNIMEM's behavior in two additional domains: congressional voting records and terrorist events. Another important issue concerning a system's behavior is how it responds to changes in parameter values. Thus we conducted a set of experiments in parameter variation, which we also present in this section.

3.1 Congressional voting records

One domain on which we tested UNIMEM involved Congressional voting records. Instances were formed from the votes of each U.S. Congressman on a number of major issues (taken from *The 1983 American Political Almanac*) combined with information about the district and state represented. One advantage of this domain for research purposes is that people have strong intuitions about the kinds of generalizations that should be found. A complete description of the domain can be found in Lebowitz (1986c). In the run described here, we presented UNIMEM with 100 instances, each containing 15 votes and about 21 other features.[18] We expected to find generalizations that related the various votes to each other (e.g., 'liberal' and 'conservative' ideologies), along with others that related the votes to the states and districts represented (e.g., someone representing a highly urban state would support bills that help cities). Indeed UNIMEM formed concepts of this sort.

Figure 3 shows several of the generalizations that resulted from this run, along with their organization in memory. One top-level generalization, GND2, describes congressmen from agricultural states with high levels of

[18]For some instances, certain features were unavailable.

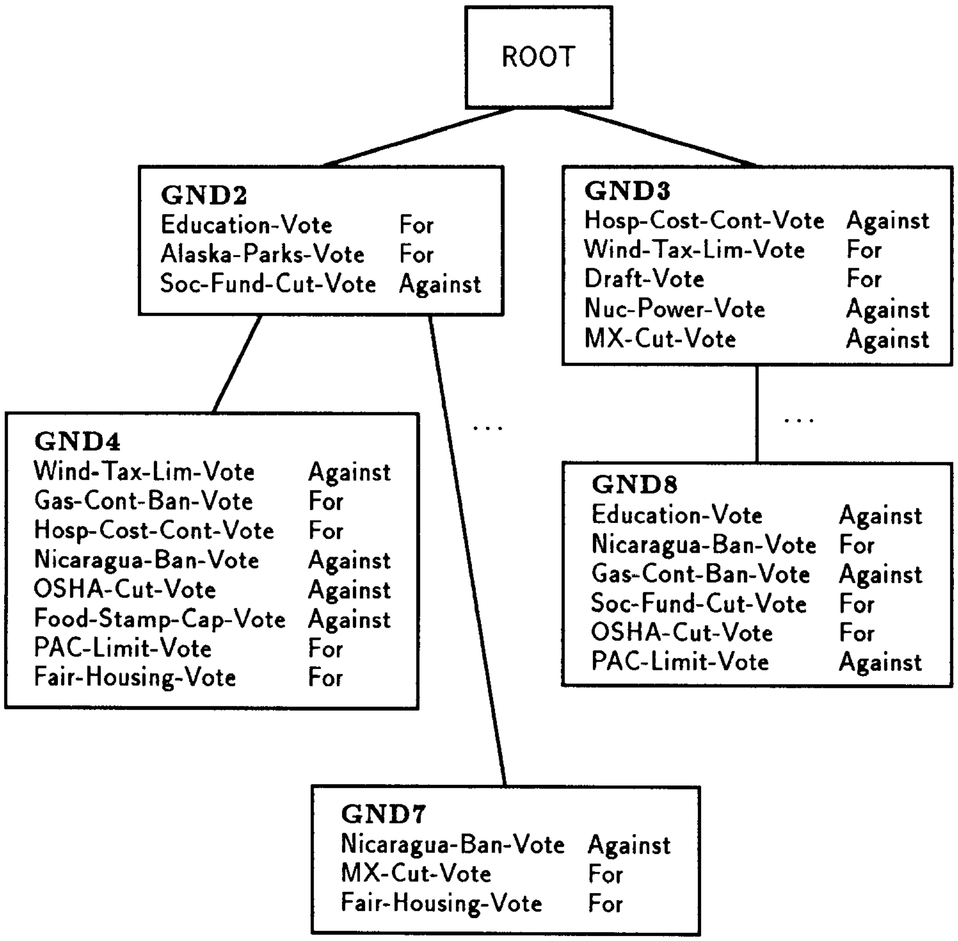

Figure 3. Partial concept hierarchy for congressional districts. Only vote-related features are shown.

school expenditures[19] who voted for an education bill, parks in Alaska, and so forth. The 24th Texas Congressional District is stored under this generalization, along with two sub-generalizations. Someone familiar with U.S. politics would describe this voting pattern as 'liberal.' Similarly, the second top-level node in this example, GND3, would be considered 'conservative.'

These two generalizations are non-disjoint, since their features do not in-

[19]Values of the form 'n out of m' represent categorized numeric information. In this domain, such categories were automatically created using methods described in Lebowitz (1985).

clude opposite votes on the same bills. Instead, the generalizations include votes on different bills and are not exclusive. A conservative record can most confidently be identified based on the votes shown in GND3, such as a vote against cutting the MX missile, while a liberal record shows up from the votes in GND2, such as a positive education vote. A Congressman can fit into both categories (indeed, this happens in the sub-generalizations of GND2). Apparently 'liberal' does not equal 'not conservative.'

The situation becomes particularly interesting when we look at the sub-generalizations of GND2 (GND4 and GND7) and GND3 (GND8). When we examine these generalizations carefully, we see that the contrasting votes omitted from the top-level generalizations appear in their sub-generalizations. For example, the 'liberal' generalization (GND2) contains a vote against a cut in social funds. The converse of this vote does not appear in GND3, but it is present in its sub-generalization, GND8. Similarly, the opposite of the conservative vote against the MX missile is not included in GND2, but it does occur in one of the sub-generalizations, GND7. Certain votes that do not serve well to define concepts at the top level can be useful in refining these concepts after the initial set of features is 'factored out.'

3.2 Terrorist events

UNIMEM was developed from the memory and generalization module of IPP (Lebowitz, 1980, 1983b, 1983c), a program that read news stories about international terrorism and added them to long-term memory. In the process, it formed a generalization hierarchy using a learning module that we will refer to as IPP-MEM. An interesting aspect of this domain is that descriptions of events tend to be incomplete, so that the instances do not have the same feature sets. UNIMEM differs from IPP-MEM in a number of technical ways. For example, parameters have been added to make it more flexible and different methods of low-level indexing are available. The most substantial change is the modification of confidence methods to consider each feature in a generalization separately. IPP-MEM maintained a single confidence level for each generalization. As a result, even one anomalous feature could cause an entire generalization to be deleted. We wanted to see whether this change in UNIMEM would dramatically alter the kinds of generalizations that remain in the generalization hierarchy at the end of a run.

The experiment described here used 374 of the stories for which the IPP text processor (IPP-NLP) produced accurate representations, all taken from the period of 1979–1980. Table 5 shows three successively more specific generalizations that UNIMEM built up from a number of bombing stories in the sample set. The features in the table with 'deleted' in their

Table 5. IPP-NLP/UNIMEM generalizations from the terrorist domain.

Attribute	*Value*	*Feature Frequency*	*Confidence Level*
GND10			[4 instances]
Weapon-Weapon	Bomb	1	6.50
Weapon-Class	Explosive	1	15.00
Results	Hurt-Person	2	15.00
Location-Area	Western-Europe	2	15.00
Methods	Explode-Bomb		deleted
S-MOP	Destructive-Attack		deleted
Victim-Nationality	Spain		deleted
Location-Nation	Spain		deleted
GND30, a more specific version of GND10			[12 instances]
Results-Health	−10	1	15.00
S-MOP	Destructive-Attack	4	11.00
Methods	Explode-Bomb	4	9.75
Victim-Nationality	Spain		deleted
GND37, a more specific version of GND30			[4 instances]
Victim-Role	Authority	1	3.75
Victim-Role	Soldier	1	−0.75
Victim-Auth	T	1	3.75
Victim-Pol-Pos	Estab	1	3.75
Victim-Nationality	England		deleted
Location-Nation	N-Ireland		deleted

confidence fields have been removed and are not part of the final generalizations (but were initially included). GND10 describes terrorist events involving bombs in Western Europe in which people were hurt. This generalization was originally formed from stories originating in Spain with an explosion taking place. While this made the generalization carry more information than the final version, it was also less widely applicable. Since other stories were found with the same characteristics, but not occurring in Spain, UNIMEM removed the location from the generalization. This allowed it to apply to a wider range of examples. Ultimately UNIMEM created a sub-generalization of GND10 (GND30) that described events in which an explosion took place and people were killed (as indicated by the −10 health value). The system also formed an even more specific variant of GND10 (GND37) in which the victims were soldiers.

The output in Table 5 is quite typical of the performance of UNIMEM in the terrorist event domain. The system created concepts that seemed to capture basic regularities in the domain. Qualitative comparison of the UNIMEM generalizations in the terrorist event domain with those generated by IPP-MEM was quite informative. Overall, the UNIMEM generalizations seemed more intuitively plausible and covered a wider range of concepts. On the other hand, they also seemed more 'bland,' omitting some of the most 'interesting' generalizations that the original system had made – for example, that pistols with silencers were frequently used in attacks on Italian political figures.

It is clear from Table 5 why the UNIMEM generalizations were more 'bland' than those of IPP. Suppose that each system formed a complicated generalization, like the one above, by noticing similar events. In response to future data, IPP-MEM would either keep the description *in toto* or delete it entirely. On the other hand, UNIMEM would inevitably refine the generalization, and make it less unusual, by removing the coincidental elements so that it covers a wider range of events. While this is mildly disappointing in the short run, overall it is quite positive. UNIMEM produces the basic generalizations (e.g., terrorist shootings usually hurt people) needed for default reasoning. Furthermore, the 'flashy' generalizations need not be lost, as they can be formed as sub-generalizations. This did not happen very often in our experiment with the terrorist domain, since there were not enough examples and, more importantly, many of the examples had very few features. Large numbers of features actually hindered IPP-MEM, as it had no way to refine over-generalized concepts. Given UNIMEM's ability to deal with greater numbers of features, we plan to increase the level of detail of the feature sets produced from IPP-NLP representations.

3.3 The effect of varying UNIMEM parameters

UNIMEM has a number of adjustable parameters that affect its behavior. Given different parameter settings, the same sequence of instances can lead to many different generalization hierarchies. In order to generate the 'best' hierarchy, we will have to find appropriate parameter settings, which may vary among domains or applications. For example, one might aim for generalizations that predict a great deal in a limited number of situations, or for ones that are widely applicable but predict only a small amount of information.[20] Convergence rate is also an issue for an incremental system like UNIMEM. Depending on the degree of consistency in the domain in

[20]Gluck and Corter (1985) and Fisher (1987) have argued on information-theoretic grounds that there is an optimal level of classification. However, their work does not apply directly to non-disjoint categories, nor to situations in which the input is uncertain and incomplete.

question, one may have to trade off learning speed with various aspects of hierarchy quality.

In order to better understand the effect of UNIMEM's parameters on the shape of the hierarchy created and on its convergence behavior, we conducted a series of experiments involving parameter variation, which we describe in this section.

3.3.1 Evaluating UNIMEM's behavior

In order to intelligently evaluate the output of UNIMEM, we must consider what makes one set of concepts better than another. We can apply the criteria recursively so that they apply to entire hierarchies. Other things being equal, we would prefer concept descriptions with many features, since each additional feature adds inferential power to the generalization. However, the more specific a generalization, the fewer examples it can be expected to cover. Thus, there is an inherent trade-off in concept formation between coverage and the ability to make predictions based on the generalizations.

A second trade-off in concept formation involves non-disjoint concepts. As pointed out earlier, allowing overlap will often result in more specific generalizations with more inferential power. However, overlap can also make the concepts less useful for a performance element, as it will have to consider how to deal with the case where a new example fits into several categories. In addition, if there are two concepts that are only slightly different, since many of the same instances will be stored under both, UNIMEM will create very similar trees of sub-generalizations, which is inefficient in both space and time.

The trade-offs between concept specificity and both generality and minimal overlap can be instantiated in UNIMEM terms with two criteria. First, under any given generalization, there should be a 'modest number' of sub-generalizations. A number in the 4–12 range seems appropriate in our domains as it yields generalizations that are relatively specific, but general enough to cover a range of instances. Second, the instances covered by a set of concepts should be divided roughly equally among them, guaranteeing that each generalization describes a number of different instances and tending to minimize overlap.

Since UNIMEM forms concepts incrementally, we must also deal with convergence. It is important to look at the time it takes the program to settle on a set of high confidence concepts that it is not likely to invalidate later in the run. Although we would like the generalization hierarchy to converge as rapidly as possible, as we will see below, this goal may conflict with the other desired properties. However, we must make sure that the program does not simply continually create and invalidate concepts.

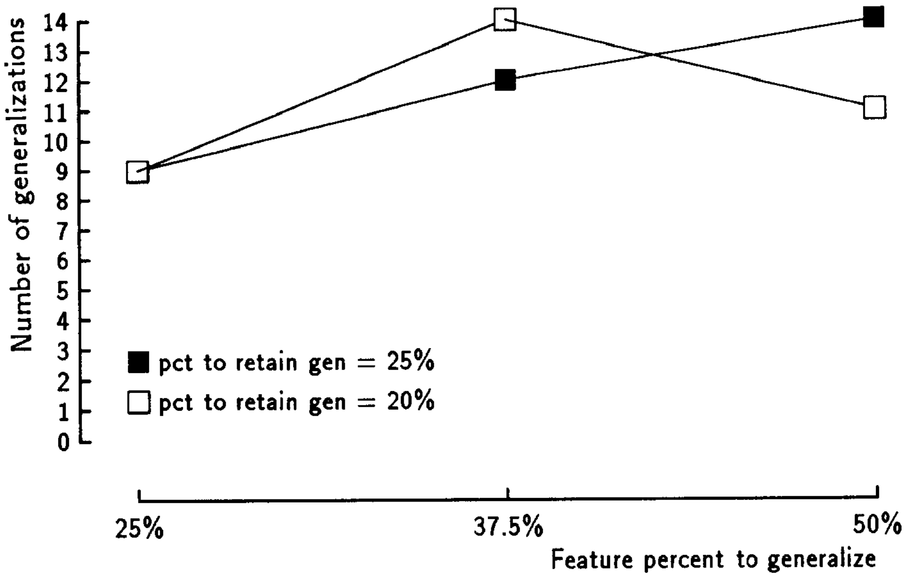

Figure 4. Generalizations at end of run as a function of percentage to generalize and percentage to retain.

3.3.2 An experiment in parameter variation

Our initial experiment involved the variation of two parameters – the percentage of features that instances must have in common for generalization to occur and the percentage of features that must remain in a generalization for it to be retained.[21] Specifically, we set the 'percentage to generalize' parameter value at 25%, 37.5%, and 50% and the 'percentage to retain a generalization' at 20% and 25%. We expected these parameters to influence UNIMEM's rate of generalization, e.g., the larger percentage of features required for a generalization, the slower the system should generalize.

The experiment involved three randomly selected sequences of 100 universities apiece.[22] For each of the six pairs of parameter values, we had UNIMEM independently incorporate the three sets of universities into an initially empty memory and then collected summary information. All of

[21]The percentage to retain a generalization parameter is computed in terms of the initial number of features in the instances.

[22]Instances contained about 20 features in this domain, so the absolute number of features needed to retain a generalization is roughly 4 at the 20% level and 5 at the 25% level. The 25% value for the features to generalize parameter requires about 5 features, the 37.5% value requires about 8, and the 50% value about 10.

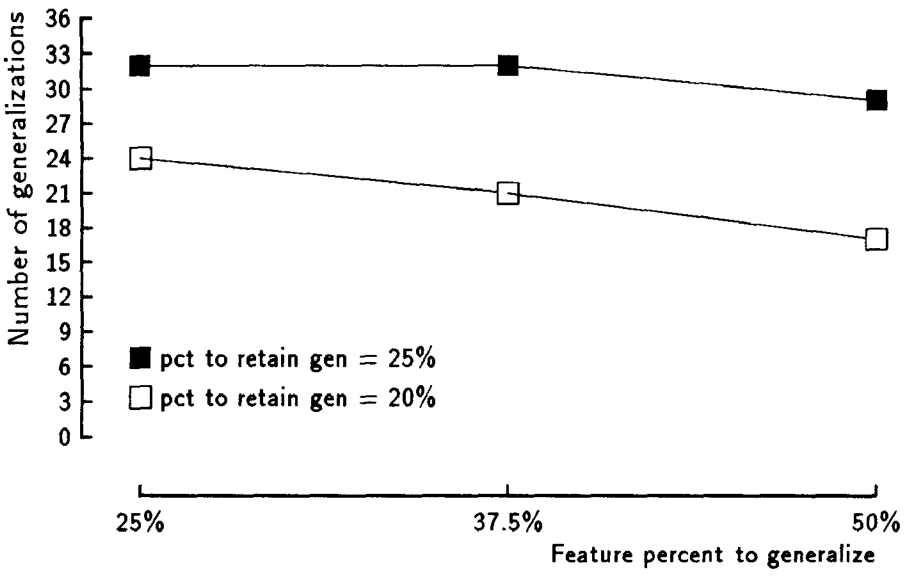

Figure 5. Generalizations created as a function of percentage to generalize and percentage to retain.

the data were averaged across the three runs.[23] Given difficulties in making assumptions about the distribution of the data, we will not present statistical analyses, but instead examine the data qualitatively. In addition, we restricted our analysis to the top-level generalizations, which can be viewed as UNIMEM's overlapping categorization of all the input instances.

The first dependent variable that we measured was the number of top-level generalizations retained by UNIMEM, as displayed in Figure 4. In the various experimental conditions the system retained an average of between 9 and 14 such generalizations, although the number will inevitably approach zero if either parameter is made very much higher. There is some indication that the number of remaining generalizations tends to increase along with each parameter, but this is not a strong trend.

In an attempt to clarify these results, we examined the two dependent variables that determine the number of generalizations that remain – the number that are created and the percentage of those created that are

[23]While UNIMEM is potentially susceptible to effects of the order of instances, this usually is not a major issue. A few odd generalizations made at the beginning of a run may have to be discarded, losing some information. In this experiment, while there was some variation in the results between the three different data sets, in no case was it striking.

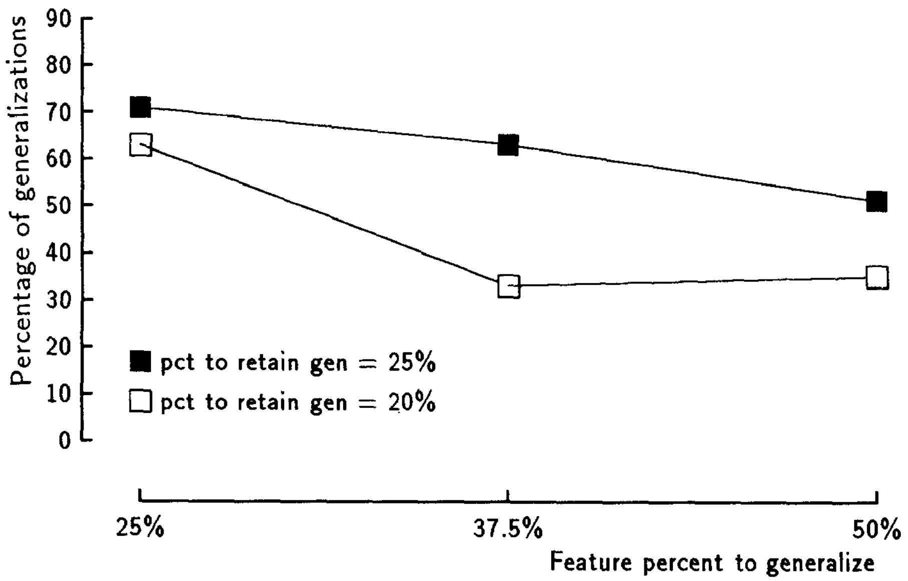

Figure 6. Generalizations deleted as a function of percentage to generalize and percentage to retain.

deleted. The average number of generalizations created for each combination of parameters is shown in Figure 5. We see, somewhat surprisingly, that the number of generalizations created declines only moderately as the features needed to generalize increases. We might expect this decline to be greater since it should be harder to find instances with more features in common. For reasons that will be considered below, the number of features needed to *retain* a generalization has a substantial effect on the number *created*.

Figure 6 shows the average percentage of generalizations deleted by UNIMEM's evaluation method when too many features were removed. As expected, more generalizations are deleted at the 25% retention level than at the 20% level. A more surprising result is that number of features needed to create a generalization affects the percentage that are deleted. The reason becomes clear when one realizes that the more features that are initially in a generalization, the more that can be removed and still be over the deletion threshold. In effect, requiring more common features to form a generalization enhances the possibility that there will be a 'good' set of features included that UNIMEM can retain once the 'bad' ones are whittled away.

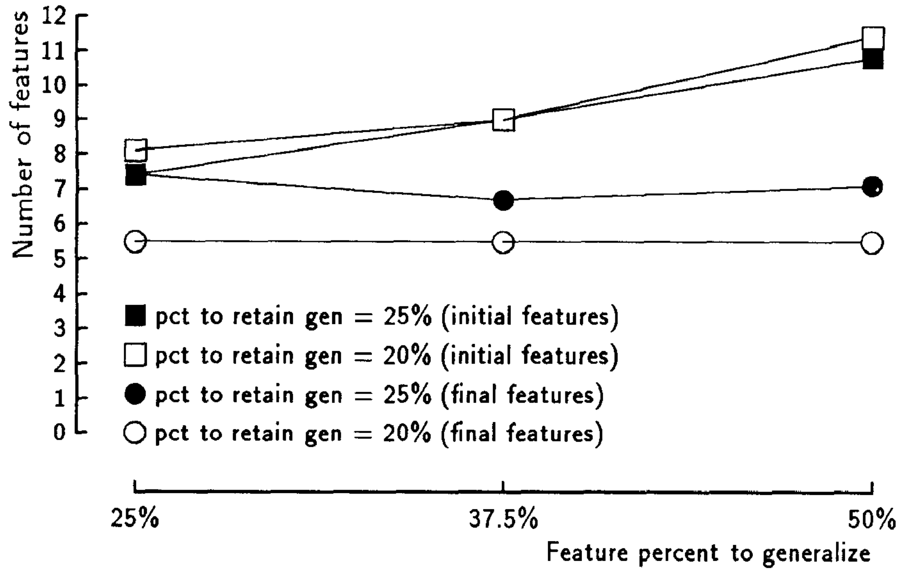

Figure 7. Average number of top-level initial and final features as a function of percentage to generalize and percentage to retain.

The decrease in the deletion rate as one increases the percentage of features needed to generalize explains the decrease in the creation rate. Since fewer generalizations are deleted, there is a higher chance that new instances will be stored under existing generalizations before the hierarchy converges. This diminishes the chance that new top-level generalizations will be created. The combination of creation and deletion behavior provides an explanation for the smaller number of generalizations retained at both ends of the 20% deletion level curve in Figure 4. If the number of features needed to generalize is very low, then few generalizations are kept, and if it is very high, then few are made. Determining the robustness of this phenomenon will require the collection of further data.

Another evaluation criterion that one might expect the parameters under consideration to influence is the average number of features in a generalization. Figure 7 shows how this variable is affected. The average final number of features that remain in each top-level generalization is essentially independent of the number of features needed to create a generalization, but it does depend upon the number needed to retain a category. The lack of any effect for the creation threshold is despite the fact that the initial number of features in a generalization, also shown in Figure 7, clearly does depend on that parameter.

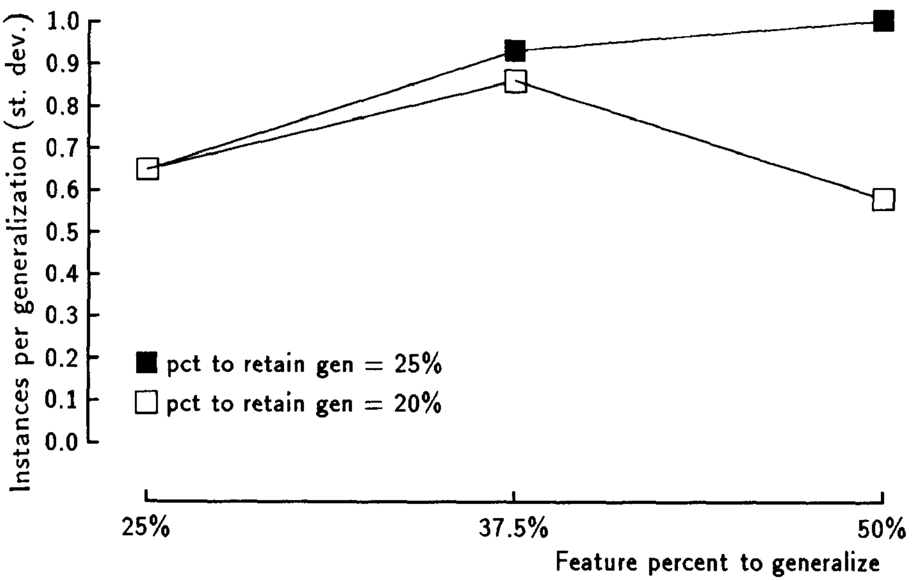

Figure 8. Average instance distribution as a function of percentage to generalize and percentage to retain.

Another one of our evaluation criteria was that, to the extent possible, instances should be evenly divided among the various concepts. To measure this, we counted the instances stored under each top-level generalization and its children. We then computed the standard deviation among the top-level generalizations, normalized in terms of the mean.[24] Figure 8 shows the results with this dependent measure. In general, the standard deviation increases (the instances are less well distributed) with the number of features needed to make a generalization. The (50%, 20%) point is a notable exception. This pair of parameter values also produced good behavior on the other measures as well. However, the results are not robust enough to draw any strong conclusions. In particular, the standard deviation results are very susceptible to new top-level generalizations created near the end of a run that describe only a small number of instances. We plan to examine this situation in more detail using longer runs and by restricting our analysis to generalizations with high-confidence features.

The final dependent variable we considered was the average feature confidence level for top-level generalizations, which we use as an approximate

[24]The normalization is needed since the total number of instances stored in the hierarchy can vary widely. This is because instances can be 'lost' when generalizations are deleted and because they can be stored in more than one place in memory.

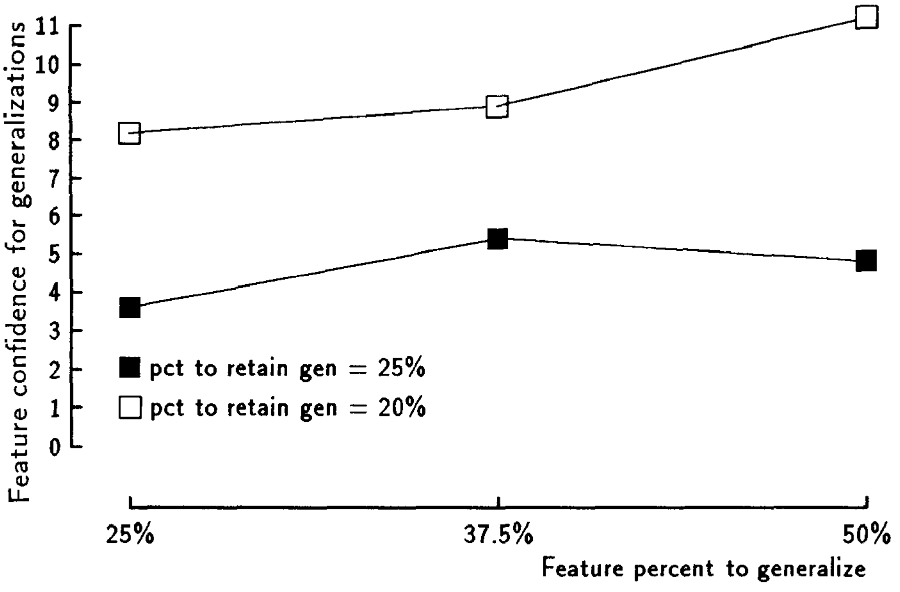

Figure 9. Average feature confidence as a function of percentage to generalize and percentage to retain.

measure of convergence. We present the results in Figure 9. The most notable thing about the data is that confidence levels are much higher for the 20% retention level than for the 25% level, and that the (50%, 20%) parameter combination clearly produces the highest confidence levels. The lower levels at the 25% retention level probably resulted from the failure of the concept set to converge before the run ended. Thus, the average reflects a number of generalizations that would ultimately have been removed entirely. We need to examine longer runs to determine whether this is strictly a convergence phenomenon or whether the confidence levels will remain lower even with a fixed set of generalizations. UNIMEM typically will not converge upon a final set of generalizations if one selects particularly poor parameter values. Whether this property is good or bad is unclear.

The results in this section illustrate the various trade-offs involving the UNIMEM parameters. Considering first the percentage to retain a generalization parameter, the lower value (20%) produced generalizations with features that had higher confidence, a desirable result that indicates more rapid convergence. However, as expected, the higher value (25%) produced generalizations with more features, which is also desirable. The results involving numbers of generalizations produced were largely inconclusive with respect to this parameter, though they did indicate that the range of values we tried produced reasonable results in terms of our evaluation criteria.

The other parameter under consideration, percentage of features needed to generalize, generally produced better results with the highest value (50%). The generalizations under that condition tended to have higher confidence features and more features and there were a reasonable number of generalizations. On the other hand, higher values of this parameter seemed to produce less well-distributed instances, although the 50% value actually produced the best result in combination with the 20% value of the 'percentage to retain' parameter. Notice that if we let the percentage to generalize parameter increase further and approach 100%, only identical instances would be used to create new concepts, which does not seem acceptable. Hence a trade-off is apparent.

It appears, then, that there is a need for intermediate values for each of the two parameters that we have examined. It seems that if instances are generalized on the basis of too few features then they are not necessarily very similar, and so their generalization has little predictive power. UNIMEM's confidence evaluation methods work well when a good generalization is embedded in the initial one, but not when the initial generalizations are essentially random. In contrast, if we require larger numbers of features to generalize than the ones used here, then the initial generalized instances have so much in common that the generalization applies to few other instances and yet appears relevant to many of them. This undermines the ability of the confidence evaluation methods to identify irrelevant features. If too many features are required to retain generalizations in relation to the number needed to make them, then almost all of the generalizations will be disconfirmed. If too few are required, then the remaining generalizations become essentially meaningless.

While collecting the data for this experiment we also saved information about the computer time needed to incorporate instances into memory. UNIMEM was designed so that the time needed to add instances to memory should increase only slightly as memory grows. Indeed, if memory reaches a point where most of the new examples are duplicates of existing ones, and hence cause no changes to the concept hierarchy, then update time should be nearly constant. In any case, the tree structure of memory should result in the time needed to update memory growing at no more than a logarithmic rate, with the growth constant depending on how efficiently instances and sub-generalizations are indexed. Figure 10 shows the empirical results for one run in the university domain, averaged over groups of ten instances.[25] The growth of update time appears consistent

[25]We average the data over groups of ten instances, since individual update times can vary radically depending upon whether a new generalization node needs to be created

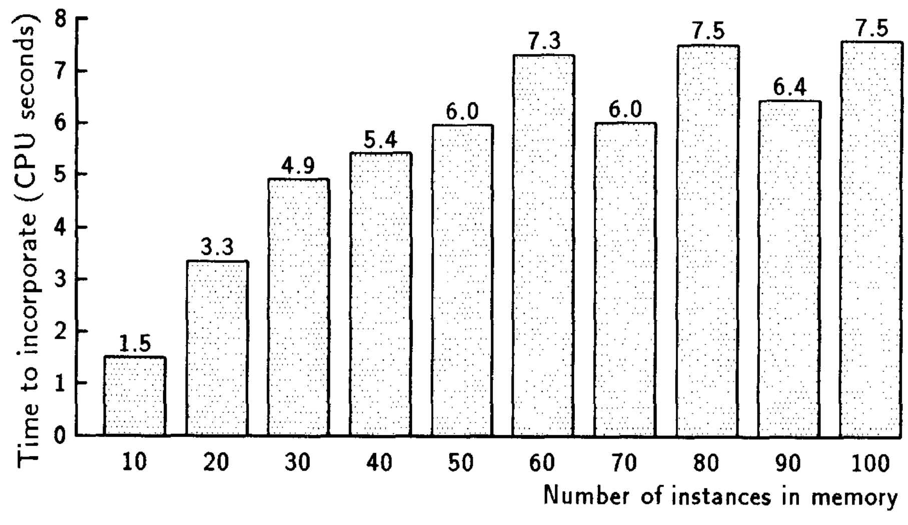

Figure 10. Time to incorporate new instances as a function of instances already in memory.

with a logarithmic increase hypothesis and it clearly does not explode in any extreme way. We plan additional experiments to better estimate the growth rate and to examine UNIMEM's behavior when greater numbers of instances are involved.

In conclusion, although we do not view the results presented herein as definitive, they have given some insight into the effects of UNIMEM parameters. In addition, we feel they show the kind of data that must be collected before we can fully understand the nature of learning by observation.

4. Related research issues

Our work with UNIMEM has left us with a number of interesting problems to pursue. We briefly describe two of them here: the automatic modification of parameter settings and the integration of explanation-based methods with the empirical approach of UNIMEM.

4.1 Automatic setting of UNIMEM parameters

We have seen that UNIMEM uses many parameters and that their settings greatly affect the system's behavior. In the long run we would like the program to set these parameters itself for each new domain. The basic

and how quickly the search through the concept hierarchy is narrowed.

idea is that UNIMEM would monitor its behavior and adjust parameters to guide it toward the desired kind of generalization hierarchy. The goal would be expressed as another set of parameters, but ones that would make intuitive sense to a user, such as the rate at which generalizations are created, the rate they are deleted, or the average branching factor of the generalization tree.

It should not be difficult to extend UNIMEM in this fashion. The system would incrementally collect data about its behavior and periodically consider adjusting its parameters in response. However, we must first understand the effects of the various parameters through experiments of the sort described above. As an initial attempt at automatic parameter adjustment we plan to have UNIMEM monitor the rate at which generalizations are deleted, and if this rate becomes too high or too low, have the system modify the parameters discussed in the previous section.

4.2 Using domain-dependent knowledge

Frequently when observing the world, humans attempt to *explain* the generalizations that they make (Schank, 1986), an ability that is lacking in UNIMEM. We are currently studying the relationship between empirical learning of the sort carried out by UNIMEM and *explanation-based* learning methods that have been developed recently (e.g., DeJong & Mooney, 1986; Mitchell, Keller, & Kedar-Cabelli, 1986; Silver, 1986). These methods, instead of looking for regularities among a large number of examples, analyze a single example in terms of cause and effect and generalize on the basis of this analysis. Roughly speaking, these methods explain an example and then generalize it, eliminating elements that are not essential to the explanation.

We have written elsewhere (Lebowitz, 1986a, 1986c) about the need to integrate these two styles of learning and our initial attempt to do so using UNIMEM. The basic approach is to explain empirically-produced generalizations using a simple rule base and to modify the UNIMEM generalizations where this is not possible. We have also begun a new project that focuses on the interaction between the two learning methods in understanding terrorism events (Danyluk, 1987).

Four assumptions underlie our plan for integrated learning. First, while an important goal of learning is indeed a causal model and many explanation-based methods consider the causality behind examples, it is often not possible to determine underlying causality. Even where this is possible, it may not be computationally practical. Second, similarity usually indicates causality and is much easier to determine, and predictability can be used to help determine the direction of causality. Third, there exist methods to refine generalizations, some of which we have seen in this paper, that mitigate the effects of coincidence. Finally, explanation-based and empirical methods complement each other effectively. In particular, it seems more efficient to use explanation-based methods to analyze generalizations rather than every individual example. Explanations can also help in deciding which empirical generalizations are likely to be significant and which features to consider.

5. Relation to other work

Our work on UNIMEM and generalization-based memory is closely related to Michalski and Stepp's (1983) research on conceptual clustering, which they developed independently at about the same time.[26] This approach also accepts feature-based instances as input and generates (from the top down) a hierarchical set of concept descriptions that summarizes them. However, the underlying mechanism is quite different from the one used by UNIMEM. For instance, Michalski and Stepp require their descriptions to perfectly describe instances they cover. In addition, their method is nonincremental, making use of an algorithm that first finds maximally general discriminants and then determines maximally specific definitions of the resulting concepts. More recent work by Stepp and Michalski (1986) makes use of domain goals to guide the search for descriptive concepts.

Fisher (1987) has developed COBWEB, a system that has much more in common with UNIMEM. This program also performs incremental concept formation, constructing a concept hierarchy from the top down to summarize instances described as sets of features. Like UNIMEM, the system fully integrates the process of recognition and learning, modifying its concept descriptions and hierarchy in the act of classifying each instance. Fisher's approach differs from the present work along a number of dimensions. While UNIMEM uses simple conjunctive definitions for concepts, COBWEB employs probabilistic representations (e.g., '60% of the instances in concept X are large and 40% are small'). In addition, the system uses an explicit evaluation function — based on Gluck and Corter's (1985) category utility metric — to determine optimal clusterings.[27] This measure would appear to be more computationally expensive than UNIMEM's matching process, but confirming this prediction would require detailed analysis. Other dif-

[26]We direct readers to Fisher and Langley (1985) for a survey of work on conceptual clustering methods.

[27]Hanson and Bauer (1986) have described WITT, another conceptual clustering system that uses an information-theoretic evaluation function. This program differs from both UNIMEM and COBWEB in that it constructs its hierarchies from the bottom up. WITT can be run in either incremental or nonincremental mode.

ferences between UNIMEM and COBWEB include the latter's requirement that categories be disjoint and support for only nominal features. However, the general approach taken in the two systems is very similar.

The approach we have taken with UNIMEM is even more closely related to Kolodner's (1984) CYRUS model. This system was initially developed at Yale contemporaneously with IPP, the predecessor of the current program.[28] Like UNIMEM, the CYRUS system builds up hierarchies of generalizations based on similarities among instances. The primary difference is that CYRUS makes much more use of domain information, which it uses to determine which elements of instances can best serve as discriminants among concepts. Kolodner's system can handle instances that contain more information than can UNIMEM, since it can apply domain knowledge to avoid combinatorial explosions in retrieval and concept formation. However, this strategy also limits its flexibility in application to new domains. Thus, UNIMEM's reliance on a relatively domain-independent feature representation can be viewed as both a strength and a weakness.

All work in conceptual clustering, including UNIMEM, can also be compared to methods for statistical clustering (e.g., Anderberg, 1973). The concept hierarchies generated by our system bear considerable resemblance to the trees produced by hierarchical statistical clustering. However, statistical approaches differ from conceptual methods in that they do not form descriptions of the resulting clusters. This results from a reliance on distance metrics to determine which clusters to form. Typically, these algorithms accept as input a matrix that specifies the distance between all pairs of instances. This may be computed from a feature-based representation or it may be given directly (e.g., by human subjects). The conceptual hierarchy is then built solely from the similarity matrix by grouping together instances that are near to each other.

This approach makes sense if one has only the distance information available, but it can lead to problems. In particular, statistical methods can cluster instances that are pairwise close, but which have no common similarities. They may also fail to cluster instances that have an underlying core of similarity, but which differ in other respects, so that they are not close together overall. If one has additional information in the form of instance descriptions, then it seems natural to take advantage of this information. This is exactly the approach taken by UNIMEM and other conceptual clustering methods. In addition, the incremental nature of UNIMEM lets it

[28]Both IPP and CYRUS were heavily influenced by Schank's (1982) theory of memory organization packets (MOPs), a psychologically-oriented theory of memory that was under development at the same time. CYRUS also involved a major effort in intelligent question answering, including reconstructing information. Section 3.2 describes how UNIMEM differs from IPP.

avoid computing distances between all pairs of instances by incorporating them into memory one at a time. Michalski and Stepp (1983) further contrast conceptual clustering with statistical methods.

6. Conclusion

Incremental concept formation is an important area of machine learning involving the automatic construction of a knowledge base that organizes real-world information. In this paper we have given an overview of UNIMEM, a program that performs concept formation incrementally. We demonstrated the system's generality by showing it in operation on several disparate domains. We also showed several of its key processes, including the creation of generalized concepts and their evaluation over time. We reported an experiment relating two of UNIMEM's parameters to the quality of the resulting set of concepts. While each new domain brings its own problems, the basic methods described here have proven to be quite robust. We feel that UNIMEM constitutes a promising step along the way toward systems that can make maximal use of information that they collect over time.

Acknowledgements

This research was supported in part by the United States Army Research Institute under contract MDA-903-85-0103. Many graduate and undergraduate students at Columbia University have been involved with UNIMEM. In particular, Ursula Wolz has contributed greatly to the development of the program. Discussions with Tom Ellman, Michelle Baker, and Andrea Danyluk have helped advance the theory. This paper is a substantially expanded version of one that appeared in the *Proceedings of the Seventh European Conference on Artificial Intelligence.* Comments by Pat Langley on earlier drafts were extremely valuable and much appreciated.

References

Anderberg, M. R. (1973). *Cluster analysis for applications.* New York: Academic Press.

Brachman, R. J. (1985). I lied about the trees. *AI Magazine*, *6*, 80–93.

Danyluk, A. P. (1987). *The use of explanations for similarity-based learning* (Technical Report). New York: Columbia University, Department of Computer Science.

DeJong, G. F., & Mooney. R. (1986). Explanation-based learning: An alternative view. *Machine Learning*, *1*, 145–176.

Dietterich, T. G., & Michalski, R. S. (1986). Learning to predict sequences. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Feigenbaum, E. A. (1963). The simulation of verbal learning behavior. In E. A. Feigenbaum & J. Feldman (Eds.), *Computers and thought*. New York: McGraw-Hill.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning, 2*, 139–172.

Fisher, D., & Langley, P. (1985). Approaches to conceptual clustering. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 691–697). Los Angeles, CA: Morgan Kaufmann.

Gluck, M., & Corter, J. (1985). Information, uncertainty, and the utility of categories. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society* (pp. 283–287). Irvine, CA: Lawrence Erlbaum.

Hanson, S. J., & Bauer, M. (1986). Conceptual clustering, semantic organization and polymorphy. *Proceedings of the International Meeting on Advances in Learning* (pp. 53–77). Les Arc, France.

Kolodner, J. L. (1984). *Retrieval and organizational strategies in conceptual memory: A computer model.* Hillsdale, NJ: Lawrence Erlbaum.

Lebowitz, M. (1980). *Generalization and memory in an integrated understanding system* (Tech. Rep. No. 186). New Haven, CT: Yale University, Department of Computer Science.

Lebowitz, M. (1983a). RESEARCHER: An overview. *Proceedings of the Third National Conference on Artificial Intelligence* (pp. 232–235). Washington, DC: Morgan Kaufmann.

Lebowitz, M. (1983b). Generalization from natural language text. *Cognitive Science. 7*, 1–40.

Lebowitz, M. (1983c). Memory-based parsing. *Artificial Intelligence, 21*, 363–404.

Lebowitz, M. (1985). Classifying numeric information for generalization. *Cognitive Science, 9*. 285–308.

Lebowitz, M. (1986a). Not the path to perdition: The utility of similarity-based learning. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 533–537). Philadelphia, PA: Morgan Kaufmann.

Lebowitz, M. (1986b). An experiment in intelligent information systems: RESEARCHER. In R. Davies (Ed.), *Intelligent library and information systems*. London: Ellis Horwood.

Lebowitz, M. (1986c). Integrated learning: Controlling explanation. *Cognitive Science, 10*, 219–240.

Michalski, R. S., & Stepp, R. E. (1983). Automated construction of classifications: Conceptual clustering versus numerical taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 5*, 396–409.

Minsky, M. (1975). A framework for representing knowledge. In P. H. Winston (Ed.), *The psychology of computer vision*. New York: McGraw-Hill.

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

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning, 1*, 47–80.

Quillian, M. R. (1968). Semantic memory. In M. Minsky (Ed.), *Semantic information processing*. Cambridge, MA: MIT Press.

Schank, R. C. (1982). *Dynamic memory: A theory of reminding and learning in computers and people*. New York: Cambridge University Press.

Schank, R. C. (1986). *Explanation patterns*. Hillsdale, NJ: Lawrence Erlbaum.

Silver B. (1986). Precondition analysis: Learning control information. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Stepp, R. E., & Michalski, R. S. (1986). Conceptual clustering: Inventing goal-oriented classifications of structured objects. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Utgoff. P. E. (1986). *Machine learning of inductive bias*. Norwell, MA: Kluwer Academic.

Winston, P. H. (1975). Learning structural descriptions from examples. In P. H. Winston (Ed.), *The psychology of computer vision*. New York: McGraw-Hill.

Appendix: UNIMEM Parameters

Below we present a complete list of the parameters used to control the UNIMEM generalization process. The only parameters that have been omitted are those that control the form of the output. Typical values are given in brackets.

- Percentage of similar instance features needed to create a generalization [40%].
- Absolute minimum number of features needed to generalize [2].
- Percentage of instance features needed to keep a generalization after some features have been deleted [20%].
- Absolute minimum number of features needed to keep a generalization [2].
- Total amount of conflict between instance features and generalization features allowed in matching [1.0].
- Confidence level at which a feature is deleted [−3].
- Confidence level at which a feature is presumed permanent [20].
- Maximum distance between feature values allowed in matching [0.5].
- Confidence multiplier for matches [2.0].
- Confidence multiplier for mismatches [−2.0].
- Number of features that indicate relevance in search [2].
- Number of misses, less than which indicates relevance [2].
- Penalty for missing feature [0.1].