INDUCTIVE LEARNING OF STRUCTURAL DESCRIPTIONS EVALUATION CRITERIA AND COMPARATIVE REVIEW OF SELECTED METHODS

by

T. G. Dietterich
R. S. Michalski

Artificial Intelligence Journal, Volume 16, No. 3, July, 1981.

Artificial Intelligence Journal,
Vol. 16, No. 3, July 1981.

Inductive Learning of Structural Descriptions:

Evaluation Criteria and Comparative Review of Selected Methods*

Thomas G. Dietterich
Computer Science Department, Stanford University, Stanford, CA 94305, U.S.A.

Ryszard S. Michalski
Department of Computer Science, University of Illinois, Urbana, ILL 61801, U.S.A.

Recommended by Bruce Buchanan

ABSTRACT
Some recent work in the area of learning structural descriptions from examples is reviewed in light of the need in many diverse disciplines for programs which can perform conceptual data analysis, i.e., can describe complex data in terms of logical, functional, and causal relationships. Traditional data analysis techniques are not adequate for discovering such relationships. Primary attention is given to methods of learning the simplest form of generalization, namely, the maximally specific conjunctive generalizations (MSC-generalizations) which completely characterize a single set of structural examples. Various important aspects of structural learning in general are examined and criteria for evaluating learning methods are presented. The criteria include the adequacy of the representation language, generalization rules used, computational efficiency, and flexibility and extensibility. Selected learning methods, developed by Buchanan et al. [2–4, 32], *Hayes-Roth* [8–11], *Vere* [34–37], *Winston* [38, 39], *and the authors, are analyzed according to these criteria. Finally some goals are suggested for future research.*

1. Introduction

1.1. Motivation and scope of paper

There are many problem areas where large volumes of data are generated about a class of objects, the behavior of a system, or a process. Scientists of

*This work was supported in part by the National Science Foundation under grants MCS-76-22940 and MCS-79-06614.

many disciplines, especially those working in experimental fields such as medicine, agriculture, chemistry, psychology, and geology, regularly face the need to analyze collections of data in order to detect regularities and common patterns. Traditional tools for data analysis include various statistical techniques, curve-fitting techniques, and numerical taxonomy. These methods, however, are often not satisfactory because they impose an overly restrictive mathematical framework on the scope of possible solutions. For example, statistical methods describe the data in terms of probability distribution functions placed on random variables. As a result, the types of patterns which they can discover are limited to those which can be expressed by placing constraints upon the parameters of various probability distribution functions. Because of the mathematical frameworks upon which they are based, these traditional methods cannot detect patterns such as the logical, causal, or functional relationships that are typical of descriptions produced by humans. This is a well-known problem in AI, namely that a system in order to learn something must first be able to express it. The solution requires introducing more powerful representations for hypotheses and developing corresponding techniques of data analysis and pattern discovery. Work done in AI and related areas on computer induction and learning structural descriptions from examples has laid the groundwork for research in this area. This is not accidental, because, as Michie [23] has pointed out, the development of systems which deal with problems in human conceptual terms is a fundamental characteristic of AI research.

This paper examines some of the work in AI on inductive learning of structural descriptions, i.e., on learning by generalizing structural descriptions representing input examples. Structural descriptions involve not only global properties of objects (represented by variables or null-ary predicates) but also properties of their parts (represented by unary functions or predicates) and relationships among these parts (represented by k-ary functions or predicates). Non-structural descriptions involve only variables and null-ary predicates. The variables, predicates, and functions involved in descriptions are called *descriptors.*

Attention is given primarily to the simplest form of inductive learning, namely that which produces the *m*aximally *s*pecific (longest) *c*onjunctive statements which characterize a given class of entities (such statements are called, for short, MSC-generalizations). These statements are an important special case of the characteristic descriptions discussed in Section 1.4. The reason for this choice is that most work done in this area is addressing this narrow, but important, subject. Many researchers have worked on other aspects of machine learning, e.g., on learning decision tree descriptions (Hunt [12], Quinlan [28, 29]), on learning recursive descriptions (Cohen and Sammut [5]), on generalization with multilevel exceptions (Vere [37]), or developing optimal discriminant descriptions of a fixed number of objects classes (Michalski [17, 20]). Work on these other subjects is much more diverse and consequently much more difficult to compare than the work on MSC-generalizations. Some of these additional contributions are mentioned in the sections concerning extensions.

This paper discusses the work of Buchanan et al. [2–4], Hayes-Roth [8–11], Vere [34–37], Winston [38, 39], and the authors' own work. In order to compare these methods, we first discuss some general aspects of inductive learning and then introduce several criteria for evaluating learning methods. These criteria are used to explain the essential contributions and ideas underlying the methods. They also make it possible to abstract from the diversity of notations and terminology used by the various authors. It is hoped that this approach will help people who are not expert in this area to better understand its progress.

Finally, some goals for future research in this area are outlined.

1.2. Important aspects of inductive learning

The process of inductive learning can be viewed as a search for plausible general descriptions (inductive assertions) which explain the given input data and are useful for predicting new data. In order for a computer program to formulate such descriptions, an appropriate description language must be used. For any set of input data and any non-trivial description language, a large number of inductive assertions can be formulated. These assertions, as noted by Mitchell [24, 25], form a set of descriptions partially ordered by the relation of generality. The minimal elements of this set are the most specific descriptions, in the given language, of the input data, and the maximal element is the most general description of these data.

Viewing induction as a search through a space of generalized descriptions draws attention to the following aspects of learning.

(a) *Representation.* What description language is used for expressing the input examples and formulating the inductive assertions? What are the possible forms of assertions which a method is able to learn? What operators are used in these forms?

(b) *Type of description sought.* For what purpose are the inductive assertions being formulated? What assumptions does the method make about the underlying process(es) which generated the data?

(c) *Rules of generalization.* What kinds of transformations are performed on the input data and intermediate descriptions in order to produce the inductive assertions?

(d) *Constructive induction.* Does the induction process change the description space, i.e., produce new descriptors which were not present in the input events?

(e) *Control strategy.* What is the strategy used to search description space: bottom-up (data-driven), top-down (model-driven), or mixed?

(f) *General versus problem-oriented methods.* Is the method oriented toward solving a general class of problems, or is it oriented toward problems in some specific application domain?

We now discuss each of these aspects in more detail.

1.3. Representation issues

There are, of course, many representational systems which can be used to represent events and generalizations of events: predicate calculus, production rules, hierarchical descriptions, semantic nets, and frames. Most work, with the exception of AM [14], has used predicate calculus (or some closely related system) because it is widely known and has a formally well-defined syntax and semantics. (An important study of theoretical problems of induction in the context of predicate calculus was undertaken by Plotkin [26, 27].)

Given a description language, learning methods using this language typically restrict the scope of the forms in which inductive assertions can be expressed. There is frequently a difference between the legal forms of the description language and the forms which a method is actually capable of learning. A useful way to characterize these 'learnable' forms is to indicate the operators which can be used in them. The most common operators are conjunction ($\wedge$), disjunction ($\vee$), internal disjunction (see below), exception, and the existential and universal quantifiers.

1.4. Types of descriptions

Since induction is a search through a description space, one must specify the goal of this search, i.e., one must provide criteria which define the goal description. These criteria depend upon the specific domain in question, but some regularities are evident. We distinguish among characteristic, discriminant, and taxonomic descriptions.

A *characteristic description* is a description of a single set of objects (examples, events) which is intended to discriminate that set of objects from all other possible objects. It is assumed that this set represents a certain conceptual class of objects. For example, a characteristic description of the set of all tables would discriminate any table from all things which are non-tables. In this way, the description *characterizes* the concept of a table. Psychologists consider this problem under the name of concept formation (e.g. Hunt [12]). Since it is impossible to examine all objects in a given class (or *not* in a given class), a characteristic description is usually developed by specifying all characteristics which are true for all *known* objects of the class (positive examples). In some problems, negative examples (counterexamples) are available which represent objects known to be not in the class. Negative examples can greatly help to circumscribe the desired conceptual class. Even more helpful are counterexamples which are 'near misses,' as demonstrated by Winston [38, 39], who uses them to create 'must not' conditions.

A *discriminant description* is a description of a class of objects in the context of a *fixed* set of other classes of objects. It states only those properties of objects in the class under consideration which are necessary to distinguish them from the objects in the other classes. A characteristic description can be viewed as a discriminant description in which the given class is discriminated against infinitely many alternative classes.

A *taxonomic description* is a description of a class of objects which subdivides the class into subclasses. In constructing such a description, it is assumed that the input data are not necessarily members of a single conceptual class. Rather it is assumed that they are members of several different classes (or produced by several different processes). An important kind of taxonomic description is a description which determines a *conceptual clustering*—a structuring of the data into object classes corresponding to certain concepts. A taxonomic description is fundamentally disjunctive. If the object classes have conjunctive descriptions, then the entire set of input data can be described as a disjunction of these descriptions.

Determination of characteristic or discriminant descriptions is the subject of learning from (pre-classified) examples, while determination of taxonomic descriptions (conceptual clustering) is the subject of what can be called learning from observation, i.e., 'learning without teacher.'

In this paper we restrict ourselves to the problem of determining characteristic descriptions. The problem of determining discriminant descriptions has been studied by Michalski and his collaborators [15–22]. A general method and computer program CLUSTER/PAF for conceptual clustering is described in [21].

1.5. Rules of generalization

The partially-ordered space of descriptions of different levels of generality can be described by indicating what transformations are being applied to change less general descriptions into more general ones. Consequently, determination of inductive assertions can be viewed as a process of consecutive application of certain 'generalization rules' to initial and intermediate descriptions. A generalization rule is a transformation rule which, when applied to an expression S_1 in the description language L, produces a more general expression S_2. This means that the implication $S_2 \Rightarrow S_1$ holds. A generalization rule is called *non-constructive* if S_2 involves no descriptors other than those used in S_1. If S_2 does contain new descriptors, then the rule is called *constructive* (see Section 1.6). Non-constructive rules of generalization do not change the representation space of the problem, while constructive rules do change it.

The concept of rules of generalization provides further insight into the view of induction as a heuristic search of description space. The rules of generalization specify the operators which the search uses to move from one node to another in this space. The concept of generalization rules is also useful for comparing different learning methods because these rules abstract from the particular description languages used in the methods.

Below are listed (non-constructive) generalization rules which are sufficient to compare the methods described in this paper. They are based on the paper by Michalski [20]. Other rules of generalization are possible, for example, rules which use quantifiers (see [22]). The rules are expressed in the first-order predicate calculus. (The rules could also be expressed in any other language with the requisite operators.) These generalizations are not necessarily *plausible*. They are merely logically possible. The notation $A \mid\!\langle B$ indicates that *A can be generalized to B*.

(i) *Dropping condition rule.* A description which uses the conjunction operator can be generalized by dropping one or more terms in the conjunction. For example

$$\text{red}(v) \wedge \text{big}(v) \;\mid\!\langle\; \text{red}(v)$$

(this reads: "the description 'vs which are red and big' *can be generalized to* the description 'vs which are red'").

(ii) *Turning constants to variables rule.* A description which has a constant as an argument can be generalized by changing the constant to a variable. It is assumed that this variable stands for (will match) any constant. (When typed variables are used, it is assumed that the variable will only match constants from the appropriate value set.) For example,

$$\text{tall(Fred)} \wedge \text{man(Fred)} \;\mid\!\langle\; \text{tall}(v) \wedge \text{man}(v)$$

These first two rules of generalization are the rules most commonly used in the literature on computer induction (e.g. by Vere [34–37], Winston [38, 39], Hayes-Roth [8–11], Hunt [12], and Fikes et al. [7]). Both rules can, however, be viewed as special cases of the following rule.

(iii) *Generalizing by internal disjunction rule.* A description can be generalized by extending the set of values which a descriptor (i.e. variable, function, or predicate) is permitted to take on in order for the description to be satisfied. This process involves an operation called the *internal disjunction*. For example

$$\text{shape}(v) = \text{square} \;\mid\!\langle\; \text{shape}(v) \in \{\text{square, triangle, rectangle}\}$$

Using the notation of variable-valued logic system VL_{21} [20] this rule can be expressed somewhat more compactly:

$$[\text{shape(v)} = \text{square}] \;\mid\!\langle\; [\text{shape(v)} = \text{square, triangle, rectangle}]$$

The commas in the expression on the right of the $\mid\!\langle$ denote the *internal disjunction*. Although it may seem at first glance that the internal disjunction is just a notational abbreviation, this operation appears to be one of the fundamental operations people use in generalizing descriptions. Its basic effect is to expand the set of values which a given descriptor is allowed to assume.

In general this rule can be expressed by

$$W[L = R1] \; |\!\langle \; W[L = R2]$$

where W is some condition (possibly empty) and $R1 \subseteq R2$. (The concatenation of W with the relational statement in brackets is interpreted as logical conjunction.)

Some work on induction [2–4, 12, 17–22] makes use of typed descriptors with specified value sets (also called *domains*). Typed descriptors provide an important mechanism for introducing domain knowledge into the learning program. Two significant special cases of the *internal disjunction* rule involve typed descriptors. First, when the descriptor takes on values which are linearly ordered (a *linear descriptor*), and the second when the descriptor takes on values which represent concepts which are partially ordered at various levels of generality (a *structured descriptor*). These generalization rules can take advantage of the properties which are known to be true of the value sets of the involved descriptor. Thus, they are more than purely logical rules of generalization; they incorporate some aspects of human plausible reasoning.

In the case of a linear descriptor we have the following rule.

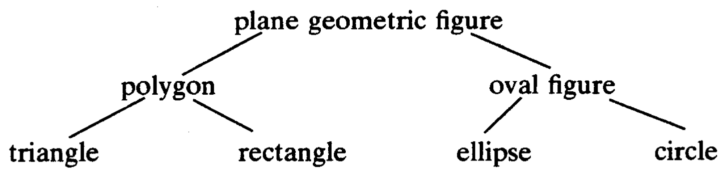
(iv) *Closing interval rule.* For example, suppose two objects of the same class have all the same characteristics except that they have different sizes, *a* and *b*. Then it is plausible to hypothesize that all objects which share these characteristics but which have sizes between *a* and *b* are also in this class.

$$\left.\begin{array}{l} W[\text{size}(v1) = a] \\ W[\text{size}(v2) = b] \end{array}\right| \langle \; W[\text{size}(v) = a\,..\,b]$$

This generalization rule makes use of the knowledge that the value set of the involved descriptor (in this case *size*) is linearly ordered.

In the case of structured descriptors we have the following rule.

(v) *Climbing generalization tree rule.* Suppose the value set of the *shape* descriptor is the tree (or hierarchy) of concepts:



With this tree structure, values such as triangle and rectangle can be generalized by climbing the generalization tree:

$$\left.\begin{array}{l} [\text{shape}(v) = \text{rectangle}] \\ [\text{shape}(v) = \text{triangle}] \end{array}\right| \langle \; [\text{shape}(v) = \text{polygon}]$$

During the generalization process, a current description may become over-generalized. If negative examples are known, then such an over-generalization will be indicated by the fact that some negative examples will satisfy the current description. To ensure that the description doesn't cover these negative examples, the description must be specialized. This can be done by applying a *specialization rule* to the current description. A reversal of any of the above generalization rules (i.e. reading the rules from right to left) is a specialization rule. Specialization rules obtained in this way can generally produce many specialized descriptions from one over-generalized description. A useful technique for avoiding the proliferation of such possible specializations is to take advantage of the negative examples which incorrectly satisfy the over-generalized description. One important way to do this is to use the following specialization rule.

(vi) *Introducing exception specialization rule.* Given a description and a negative example (incorrectly) satisfying it, this rule creates an exception condition and adds it to the initial description. The general form of the rule is:

$$\left.\begin{array}{ll}\text{current description:} & P(v)\\ \text{negative example:} & P(v)\wedge Q(v)\end{array}\right| \rangle\ P(v)\setminus Q'(v)$$

Where \ denotes the exception operator. The expression $P(v)\setminus Q'(v)$ is read as "$P(v)$ except when $Q'(v)$" and is logically equivalent to $P(v)\wedge\neg Q'(v)$. $Q'(v)$ is either the same description as $Q(v)$ or else a generalization of $Q(v)$. The symbol $|\rangle$ denotes specialization.

As an example, consider the problem of learning the concept of "fish".

$$\left.\begin{array}{ll}\text{current description:} & \text{swims}(v)\\ \text{negative example:} & \text{swims}(v)\wedge\text{breathes-air}(v)\end{array}\right| \rangle\ \begin{array}{l}\text{swims}(v)\setminus\\ \quad\text{breathes-air}(v)\end{array}$$

It is easy to see that the introducing exception specialization rule is a special case of the 'adding condition specialization rule' (i.e. the inverse of the dropping condition generalization rule).

1.6. Constructive induction

All of the generalization and specialization rules presented above involve the same descriptors on both sides of the rule. Most methods of induction (e.g. by Hunt [12], Hayes-Roth [8–11], Vere [34, 37], Mitchell [24–25]) use only these kinds of generalization rules and consequently produce descriptions which involve the same descriptors which were present in the initial data. Such methods perform *non-constructive* induction. A method performs *constructive* induction if it includes mechanisms which can generate new descriptors not present in the input data. Constructive induction is a general term which describes *any* induction technique which produces such new descriptors. A program which performs constructive induction makes changes to the representation of a problem. It is well known that in many AI problems (for example, the mutilated checkerboard problem) finding an appropriate problem representation is crucial to finding a good solution.

We find it useful to classify constructive induction techniques in terms of rules for constructive induction. Constructive induction rules are stated independently of their implementation. They may be based on general knowledge or on problem-specific knowledge. Some examples of constructive generalization rules are presented in [20, 22]. Rules of constructive induction can be implemented as programs which generate new descriptors as functions of the initial descriptors, and are invoked when certain contextual conditions are satisfied. For example, when a program determines that parts of objects constitute a linearly ordered set, then a procedure can be invoked to generate various descriptors which apply to linearly ordered sets (e.g., the first element, the last element, the length). The 'grouping with a typical element' transformation described by Winston [38, 39] is another example of constructive induction.

One of our goals for further research is the identification and implementation of constructive induction rules. An inductive program should contain facilities for constructive induction including a library of general constructive induction rules. The user should be able to suggest new rules for the program to apply. As more sophisticated constructive induction rules are implemented, the ability of inductive programs to accomplish radical representation change should be enhanced. However, the increased number of rules will also lead to increased search problems. Consequently, these learning programs will also need heuristic meta-rules which can guide the process of evoking constructive induction rules.

1.7. Control strategy

Methods can be divided into bottom-up (data-driven), top-down (model-driven), and mixed methods. Bottom-up methods generalize the input events pairwise until the final conjunctive generalization is computed:

$$\begin{tikzcd}
 & & & G_4 \\
 & & G_3 \arrow[ur, no head] & \\
 & G_2 \arrow[ur, no head] & & \\
E_1 = G_1 \arrow[ur, no head] & E_2 \arrow[u, no head] & E_3 \arrow[uu, no head] & E_4 \arrow[uuu, no head]
\end{tikzcd}$$

G_2 is the set of conjunctive generalizations of E_1 and E_2. G_i is the set of conjunctive generalizations obtained by taking each element of G_{i-1} and generalizing it with E_i.

Only the methods described by Winston, Hayes-Roth, and Vere are reviewed in this paper. Some other bottom-up methods include the candidate elimination approach described by Mitchell [24, 25], the ID3 technique of Quinlan [28, 29], and the Uniclass method described by Stepp [33].

Model-driven methods search a set of possible generalizations in an attempt to find a few 'best' hypotheses which satisfy certain requirements. The two methods discussed in this paper (by Buchanan et al. and Michalski) search for a small number of conjunctions which together cover all of the input events. The search proceeds by choosing as the initial working hypothesis some starting point in the partially ordered set of all possible descriptions. If the working hypotheses satisfy certain termination criteria, then the search halts. Otherwise, the current hypotheses are modified by slightly generalizing or specializing them. These new hypotheses are then checked to see if they satisfy the termination criteria. The process of modifying and checking continues until the criteria are met. Top-down techniques typically have better noise immunity, and can easily be extended to discover disjunctions. The principal disadvantage of these techniques is that the working hypotheses must repeatedly be checked to determine whether they subsume all of the input events.

1.8. General versus problem-oriented methods

It is a common view that general methods of induction, although mathematically elegant and theoretically applicable to many problems, are in practice very inefficient and rarely lead to any interesting solutions. This opinion seems to have lead certain workers to abandon (at least temporarily) work on general methods and concentrate on learning problems in some specific domains (e.g., Buchanan et al. [2–4] or Lenat [14]). Such an approach can produce novel and practical solutions. On the other hand, it is difficult to extract general principles of induction from such problem-specific work. It is also difficult to apply such special-purpose programs to new areas.

An attractive possibility for solving this dilemma is to develop methods which incorporate various general principles of induction (including constructive induction) together with mechanisms for using exchangeable packages of problem-specific knowledge. In this way a general method of induction, provided with an appropriate package of knowledge, could be both readily applicable to different problems, and also efficient and practically useful. This idea underlies the development of the INDUCE programs [15, 16, 20].

2. Comparative Review of Selected Methods

2.1. Evaluation criteria

The selected methods of induction are evaluated in terms of several criteria considered especially important in view of the remarks in Section 1.

(i) *Adequacy of the representation language.* The language used to represent input data and output generalizations determines to a large extent the quality and usefulness of the output descriptions. Although it is difficult to assess the adequacy of a representation language out of the context of some specific problem, recent work in AI has shown that languages which treat all phenomena uniformly must sacrifice descriptive precision. For example, researchers who are attempting to build systems for understanding natural language prefer the richer knowledge representations, such as frames and semantic nets (with their tremendous variety of syntactic forms), to the more uniform and less structured representations, such as attribute-value lists and PLANNER-style representations. Although languages with many syntactic forms do provide greater descriptive precision, they also lead to exponential increases in the complexity to the induction process. In order to control this complexity, a compromise must be sought between uniformity and richness of forms. In the evaluation of each method, a review of the operators and syntactic forms of each description language is provided.

(ii) *Rules of generalization implemented.* The generalization rules implemented in each algorithm are listed.

(iii) *Computational efficiency.* The exact analysis of the computational efficiency of the selected learning algorithms is very difficult due both to the inherent complexity of the algorithms and to the lack of precise formulations of the algorithms in available publications. However, it seems useful to have some data comparing the efficiency of these algorithms even if that data is based on the hand-simulation of only one sample problem. We have hand-simulated each algorithm on the test problem shown in Fig. 2. To get some indication of efficiency, we measure the total number of description generations and description comparisons required for this sample problem. The rationale for counting description comparisons is that the comparison of two structural descriptions (which are typically complex expressions or graphs) is a very expensive operation (since subgraph isomorphism is NP-complete). Description generations also provide an important measure of effort—especially for top-down algorithms which use a form of generate-and-test search. The ratio of the number of output conjunctive generalizations (i.e. the generalizations produced as the final result of the algorithm) to the total number of generalizations examined is also computed. Since these numbers are derived from only one example, it is not appropriate to draw strong conclusions from them concerning the general performance of the algorithms. Our evaluation is based primarily on the general behavior of the algorithms.

(iv) *Flexibility and extensibility.* Programs which can only discover conjunctive characteristic descriptions have very limited practical applications. In particular, they are inadequate in situations involving noisy data or in which no single conjunctive description can describe the phenomena of interest. Consequently, as one of the evaluation criteria, we consider the ease with which each method could be extended to

(a) discover descriptions with forms other than conjunctive generalizations (see Section 1.3),

(b) include mechanisms which facilitate the detection of errors in the input data,

(c) Provide a general facility for incorporating domain-specific knowledge into the induction process as an exchangeable package (ideally, the domain-specific knowledge should be isolated from the general-purpose inductive process), and

(d) perform constructive induction.

It is difficult to assess the flexibility and extensibility of the algorithms considered here. We base our evaluation on the general approaches of the methods and on extensions which have already been made to them.

In the following sections, we describe each method by presenting the description language used, sketching the underlying algorithm, and evaluating the method in terms of the above criteria. Two sample learning problems will be used to explain the methods. The first problem (Fig. 1) is made up of two examples (E1 and E2). Each example consists of objects (geometrical figures) which can be described by

- attributes *size* (small or large) and *shape* (circle or square), and
- relationships *ontop* (which indicates that one object is above another) and *inside* (which indicates that one object lies inside another).

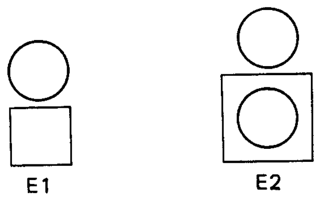


FIG. 1. Sample problem for illustrating representation languages.

The second sample problem (Fig. 2) contains three examples which are also constructions made of simple geometrical objects. These objects can be described by

- attributes *shape* (box, triangle, rectangle, ellipse, circle, square, or diamond), *size* (small, medium, or large), and *texture* (blank or shaded), and
- relationships *ontop* and *inside* (the same as in the first sample problem).

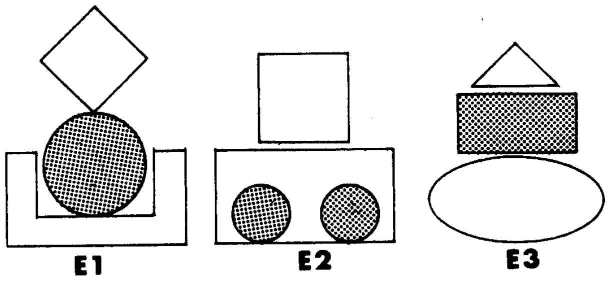


FIG. 2. Sample problem for comparing the performance of the methods.

In each sample problem, the task is to determine a set of maximally specific conjunctive generalizations (MSC-generalizations) of the examples. No negative examples are supplied in either problem. In the discussion below, the first problem is used to illustrate the representational formalism and the generalization process implemented in each method. The second, more complex, problem is used to compare the computational efficiency and representational adequacy of each method. This comparison is based on the hand simulation of each method as applied to this problem.

2.2. Data-driven methods: Winston, Hayes-Roth, and Vere

2.2.1. *Winston*: *Learning blocks world concepts* [38, 39]

Winston's well-known work [38, 39] deals with learning concepts which characterize simple toy block constructions. Although his method uses no precise criterion to define the goal description, the method usually develops MSC-generalizations of the input examples. The method assumes that the examples are provided to the program by an intelligent teacher who carefully chooses both the kinds of examples used and their order of presentation. The program uses so-called 'near-miss' negative examples to rapidly determine the correct generalized description of the concept. The near-misses are also used to develop 'emphatic' conditions such as 'must support' or 'must not support.' These *Must-* type descriptors indicate which conditions in the concept description are necessary to eliminate negative examples.

As Knapman has pointed out in his review of Winston's work [13], many parts of the exposition in Winston's thesis [38] and subsequent publication [39] are not entirely clear. Although the general ideas in the thesis are well explained, the exact implementation of these ideas is difficult to extract from these publications. Consequently, our description of Winston's method is necessarily a reconstruction.

The method uses a semantic network to represent the input events, the background blocks-world knowledge, and the concept descriptions generated by the program. The representation is quite general, although the implemented programs appear to process the network in domain-specific ways (see [13], [38, p. 196]). The algorithms which convert line drawings into the network representation and perform object recognition use knowledge specific to the blocks world.

Fig. 3 shows the network representation of the two examples in Fig. 1.

Nodes in the network are used for several different purposes. We will illustrate these purposes by referring to the corresponding concepts in first-order predicate logic. The first use of nodes is to represent various primitive concepts which are properties of objects or their parts (e.g., *small, size, circle, shape*). Nodes in this case correspond to constants in first-order predicate logic expressions. There is no distinction between attributes and values of attributes

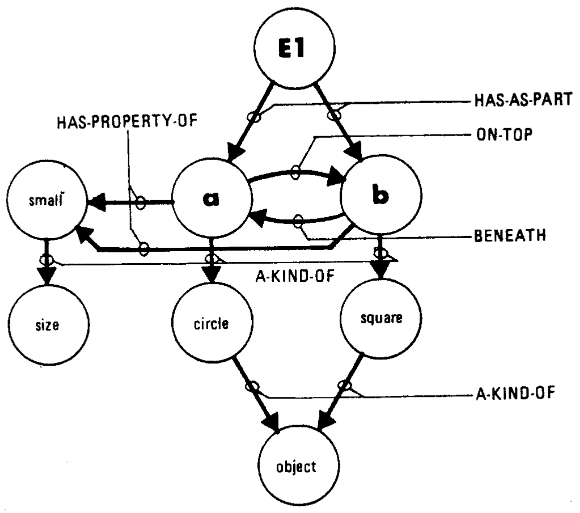


FIG. 3a. Network representing example E1 in Fig. 1.

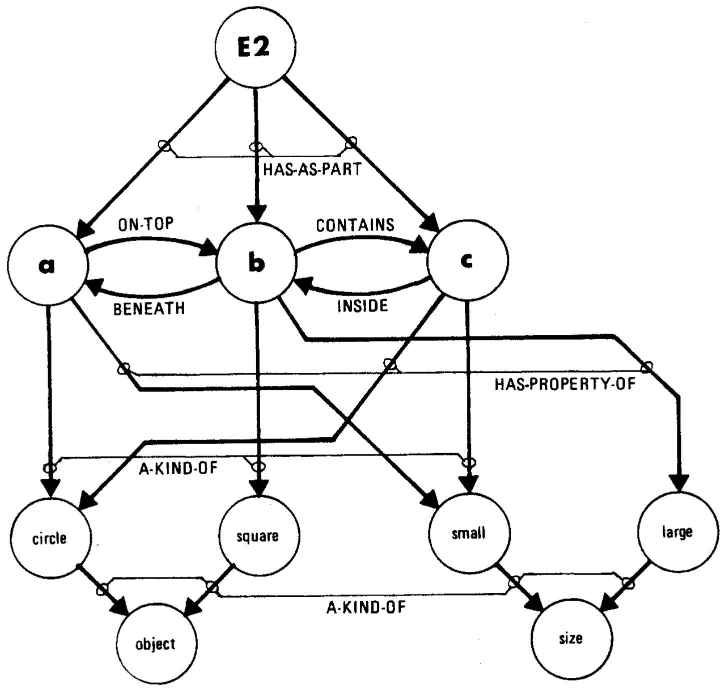


FIG. 3b. Network representing example E2 in Fig. 1.

in Winston's network representation, and, consequently, there is no representational equivalent of the one-argument predicates and functions of FOPL.

Another use of nodes is to represent individual examples and their parts. Thus, in Fig. 3a, we have the node E1 and two nodes A and B which make up E1. These can be regarded as quantified variables in predicate calculus. Distinct variable nodes are created for each training example.

Labeled links connecting these nodes represent various binary relationships among the nodes. The links correspond to two-argument predicates. The first two uses of nodes as constants and variables, plus the standard use of links as predicates, constitute the basic semantic network representation used by Winston.

There is, however, a third use of nodes. Each link type (analogous to a predicate symbol) is also represented in the network as a node. Thus, in addition to the numerous *On-Top* links which may appear in the network, there is one *On-Top* node which describes the link type *On-Top* and its relationship to other types. For example, there might be a *Negative-Satellite* link which joins the *On-Top* node to the *Beneath* node. Such a link indicates that *On-Top* and *Beneath* are semantically opposite predicates. Similarly, there is a ***Must-Be-Satellite*** link connecting the ***Must-Be-On-Top*** node to the *On-Top* node.

All of the nodes in the network are joined into one generalization hierarchy through the *A-Kind-Of* links. This hierarchy is used to implement the climbing generalization tree rule.

The learning algorithm proceeds in two steps. First, the current concept description is compared to the next example, and a difference description is developed. Then this difference description is processed to obtain a new, generalized concept description. Often, the second step results in several possible generalized concept descriptions. In such a case, one generalized concept is selected for further refinement and the remaining possibilities are placed on a backtrack list. The program backtracks when it is unable to consistently generalize its current concept description.

The first step of the algorithm (the development of the difference description) is accomplished by graph-matching the current concept description against the example supplied by the teacher, and annotating this match with comment notes (C-NOTES). These C-NOTES describe conditions in the concept description and example which partially matched or did not match. Winston's description of the graph-matching algorithm is sketchy ([13] and [38, pp. 254–263]). The algorithm apparently finds one 'best' match between the training example and the current concept description. The method does not address the important problem of multiple graph sub-isomorphisms, i.e., the problem arising when the training example matches the current concept description in more than one way. This problem was apparently avoided by assuming that the teacher will present training instances which can be unambiguously matched to the current concept description.

Once this match between the concept description and the example is obtained, a generalized skeleton is created containing only those links and nodes which matched exactly. The C-NOTES are then attached to this skeleton. Each C-NOTE is a sub-network of nodes and links which describes a particular type of match. There are several types of C-NOTES. First, there are *Intersection* C-NOTES which indicate when two nodes match exactly. Second, there are *Exit* and *Supplementary pointer* C-NOTES which handle unmatched nodes and links respectively. Then there are C-NOTES for partially matched nodes. The *A-Kind-Of-Merge* C-NOTE handles the case when two nodes mismatch, for example when *square* mismatches *triangle*. The *A-Kind-Of-Chain* C-NOTE handles the case when a node matches a more general node, for example when *square* matches *polygon*.

The remaining C-NOTES handle partially matched links. A *Negative-Satellite-Pair* C-NOTE indicates that two semantically opposite links mismatched, for example *Marries* and *Does-Not-Marry*. A *Must-Be-Satellite-Pair* C-NOTE indicates that a normal link, e.g. *Supports*, matches an emphatic link, e.g. *Must-Support*. A *Must-Not-Be-Satellite-Pair* C-NOTE indicates that a normal link matches a *Must-Not* form of the same link.

Table 1 lists the C-NOTE types according to the circumstances in which they are used. The network diagram of Fig. 4 shows the difference description which results from matching the two networks of Fig. 3 to each other.

The generalization phase of the algorithm is fairly simple. Each C-NOTE is handled in a way determined by the C-NOTE type and whether the example is an instance of the concept or a near-miss. Winston provides a table that indicates what actions his program takes in each case ([38, pp. 145–146]).

Some C-NOTES can be handled in multiple ways. For positive examples, only one C-NOTE causes problems: the *A-Kind-Of-Merge*. In this case, the program can either climb the *A-Kind-Of* generalization tree or else drop the condition altogether. The program develops both possibilities but only pursues the former (leaving the latter on the backtrack list). The concept description which results from generalizing the difference description of Fig. 4 is shown in Fig. 5.

The alternative generalization would drop the *Has-Property* link from node *b*.

TABLE 1. Winston's C-NOTE categories

	Match	Partially match	Mismatch
Node	Intersection	A-Kind-Of Merge	Exit
		A-Kind-Of Chain	
Link		Negative-Satellite-Pair	
		Must-Be-Satellite-Pair	Supplementary pointer
		Must-Not-Be-Satellite-Pair	

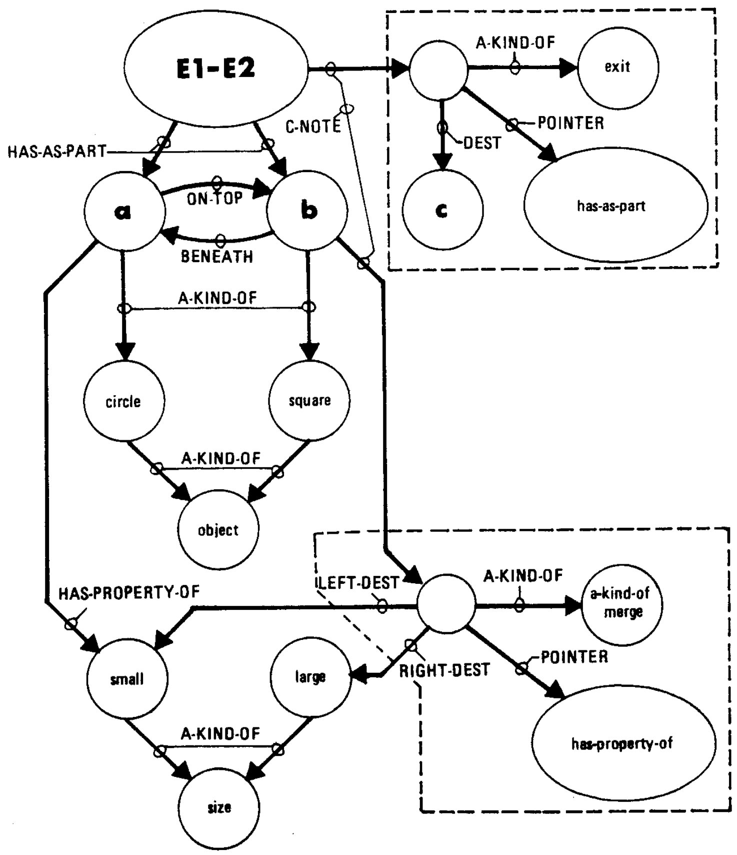


FIG. 4. Difference description obtained by comparing E1 and E2 from Fig. 1 and annotating the comparison with two C-NOTES.

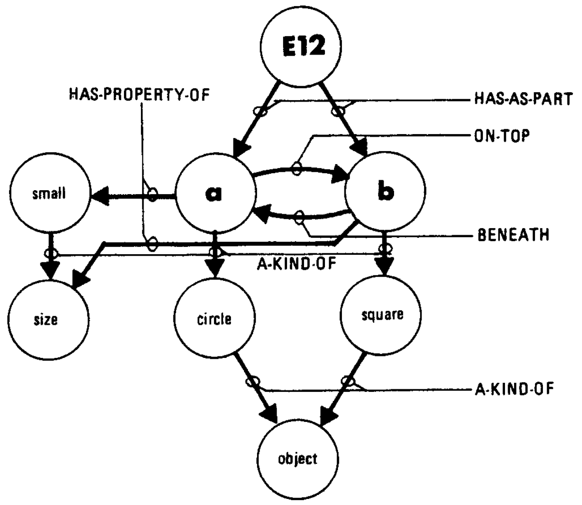


FIG. 5. Network representing the generalized concept resulting from generalizing the difference description of Fig. 4.

Evaluation. (i) ***Representational adequacy***. The semantic network is used to represent properties, object hierarchies (using *A-Kind-Of*), and binary relationships. As in most semantic networks, n-ary relationships cannot be represented directly. The conjunction operator is implicit in the structure of the network, since all of the conditions represented in the network are assumed to hold simultaneously. There is no mechanism indicated for representing disjunction or internal disjunction. The *Not* and *Must-Not* links implement a form of the exception operator. An interesting feature of Winston's work is the use of the emphatic *Must-* relationships.

The program works in a depth-first fashion and produces only one generalized concept description for any given order of the training examples. Permuting the training examples may lead to a different generalization. Two generalizations obtained by simulating Winston's learning algorithm on the examples of Fig. 2 are shown in Figs. 6a and 6b.

The second generalization (Fig. 6b) is not maximally specific since it does not mention the fact that all training examples also contain a small or medium sized shaded object. The algorithm cannot discover this generalization due to the fact that the graph-matcher finds the 'best' match of the current concept

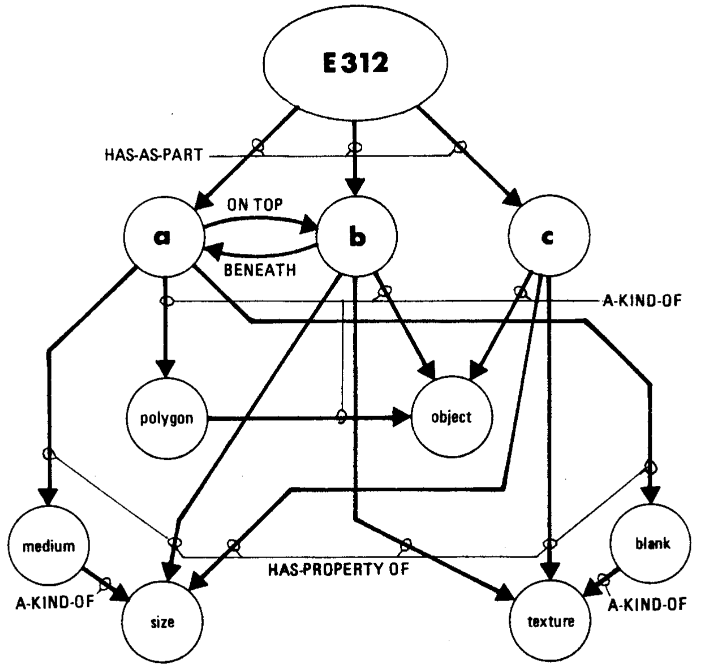


FIG. 6a. The first generalization obtained by simulating Winston's learning algorithm on the examples of Fig. 2 (in the order E3, E1, E2). An English paraphrase is: "There is a medium, blank polygon on top of another object which has a size and texture. There is also another object with size and texture."

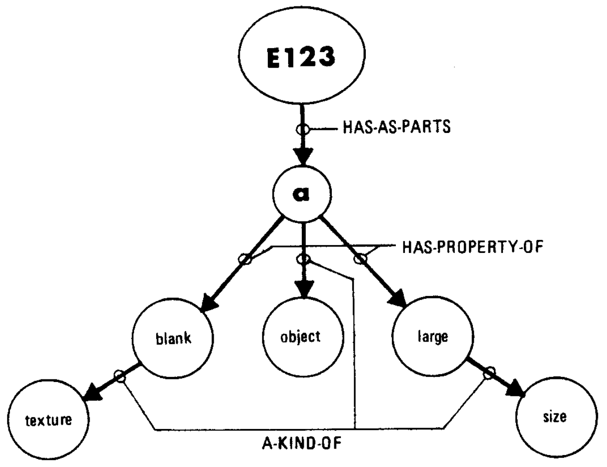


FIG. 6b. The second generalization obtained by simulating Winston's learning algorithm on the examples of Fig. 2 (in the order E1, E2, E3). An English paraphrase is: "There is a large, blank object."

with the example. When the order of presentation of the examples is E1 followed by E2 followed by E3, the 'best' match of the first two examples eliminates the possibility of discovering the maximally specific conjunctive generalization when the third example is matched.

(ii) *Rules of Generalization.* The program uses the dropping condition rule (for generalizing exit C-NOTES), the turning constants to variables rule (when creating the generalized skeleton), and the climbing generalization tree rule (for the *A-Kind-Of-Merge*). It also uses the introducing exception specialization rule (for the *A-Kind-Of-Merge* C-NOTE with negative examples).

(iii) *Computational efficiency.* The algorithm is quite fast: it requires only 2 graph comparisons to handle the examples of Fig. 2. However, the algorithm does use a lot of memory to store intermediate descriptions. The first graph comparison produces 8 alternatives of which only one is pursued. The second graph comparison leads to 4 more alternatives from which one is selected as the 'best' concept description. This inefficient use of memory is reflected in our figure for computational efficiency (the number of output descriptions / the number of examined descriptions) which is 1/11 or 9%.

The performance of the algorithm can be much worse in certain situations. When 'poor' negative examples are used—those which do not match the current concept description well—the number of intermediate descriptions explodes combinatorially. Such situations are also likely to cause extensive backtracking.

Since the algorithm produces only one generalization for any given order of the input examples, it must be executed repeatedly if several alternative generalizations are desired.

(iv) *Flexibility and Extensibility*. It seems that it would be difficult to extend the method to produce disjunctive concepts. The algorithm operates under assumption that there is one conjunctive concept characterizing the examples, so the development of disjunctive concepts is not consistent with the spirit of the work.

Since the program behaves in a depth-first manner, noisy training events cause it to make serious errors from which it cannot recover without extensive backtracking. This is not surprising since Winston assumes that the teacher is intelligent and does not make any mistakes in training the student. It seems to be very difficult to extend this method to handle noisy input data.

The inductive generalization portion of the program does not contain much problem-specific knowledge. However, many of the techniques used in the program, e.g., building complete difference descriptions and using a backtracking search, would not be feasible in any real-world problem domain. The *A-Kind-Of* generalization hierarchy can be used to represent problem-specific knowledge.

The system of programs described by Winston performs some types of constructive induction. The original inputs to the system are noise-free line drawings. Some knowledge-based algorithms convert these line drawings into the network representation. Winston describes an algorithm for combining a group of objects into a single concept and subsequently using this concept in other descriptions. The 'arcade' concept ([38, p. 183]) is a good example of such a constructive induction process.

2.2.2. *Hayes-Roth*: *Program* SPROUTER [8–11]

Hayes-Roth's work on induction [8–11] is concerned with finding MSC-generalizations of a set of input positive examples (in his work such generalizations are called *maximal abstractions* or *interference matches*). *Parameterized structural representations* (PSR's) are used to represent both the input events and their generalizations. The PSR's for the two events of Fig. 1 are

E1: {{circle:a}{square:b}{small:a}
{small:b}{ontop:a, under:b}}

E2: {{circle:c}{square:d}{circle:e}
{small:c}{large:d}{small:e}
{ontop:c, under:d}{inside:e, outside:d}}

Expressions such as {small:a} are case frames, made up of case labels (small, circle, etc.) and parameters (a, b, c, etc.). The PSR can be interpreted as a conjunction of predicates of the form

case-label(parameter-list)

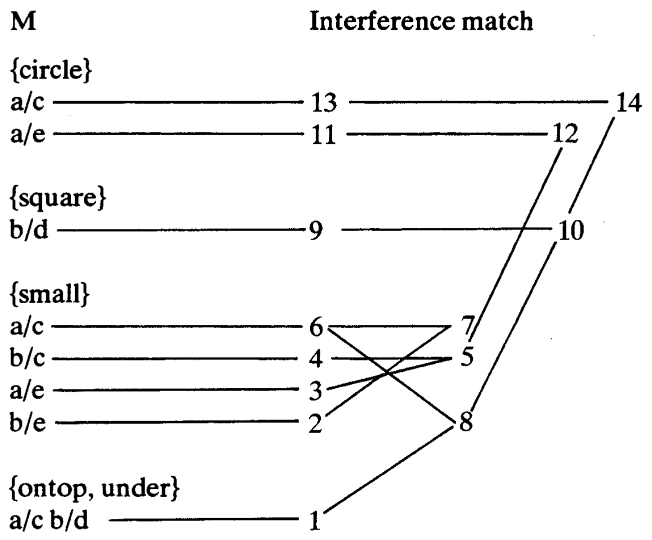
for example {small:a} can be interpreted as small(a). The parameters can be viewed as existentially quantified variables denoting distinct objects.

The induction algorithm works in a purely bottom-up fashion. The first set of conjunctive generalizations, G_1, is initialized to contain only the first input example. Given a new example and the set of generalizations, G_i, obtained in the *i*th step, a new set of generalizations, G_{i+1}, is obtained by performing an ***interference match*** between each element in G_i and the current training example. It is not clear from publications [8–11] whether or not these sets G_i are pruned during this process.

The interference match attempts to find the longest one-to-one match of parameters and case frames (i.e., the longest common subexpression). This is accomplished in two steps. First, the case frames in E1 and E2 are matched in all possible ways to obtain the set M. Two case frames match if all of their case labels match. Each element of M is a case frame and a list of parameter correspondences which permit that case frame to match in both events:

M = {{circle:((a/c)(a/e))},
{square:((b/d))},
{small:((a/c)(b/c)(a/e)(b/e))},
{ontop, under:((a/c b/d))}}

The second step involves selecting a subset of the parameter correspondences in M such that all parameters can be bound consistently. This is conducted by a breadth-first search of the space of possible bindings with pruning of unpromising nodes. The search can be visualized as a node-building process. Here is one such (pruned) search graph:



The nodes are numbered in order of their generation. One at a time, a pair of corresponding parameters is selected from M, and a new node is created for them. Then this new node is compared with all previously generated nodes.

Additional nodes are created for each case in which the new parameter correspondence node can be consistently merged with a previously existing node. In the search graph above, when the parameter binding {small:(a/c)} is selected, node 6 is created. Then node 6 is compared to nodes 1 through 5 and two new nodes are created: node 7, which is created by merging node 6 (a/c) with node 2 (b/e), and node 8, which is created by merging node 6 (a/c) with node 1 (a/c b/d). Node 6 cannot be merged with node 3, for instance, because parameter a would be inconsistently bound to both parameters c and e.

When the search is completed, nodes 7, 12, and 14 are bindings which lead to conjunctive generalizations. Node 14, for example, binds a to c (to give v1) and b to d (to give v2) to produce the conjunction:

{{circle:v1}{square:v2}{small:v1}{ontop:v1, under:v2}}

The node-building process is guided by computing a utility value for each candidate node to be built. The nodes are pruned by setting an upper limit on the total number of possible nodes and pruning nodes of low utility when that limit is reached.

Evaluation. (i) *Representational adequacy.* The algorithm discovers the following conjunctive generalizations of the example in Fig. 2.

1. {{ontop:v1, under:v2}{medium:v1}{blank:v1}}

There is a medium, blank object ontop of something.

2. {{ontop:v1, under:v2}{medium:v1}{large:v2}{blank:v2}}

There is a medium sized object ontop of a large, blank object.

3. {{medium:v1}{blank:v1}{large:v3}{blank:v3}{shaded:v2}}

There is a medium sized, blank object, a large sized, blank object, and a shaded object.

PSR's provide two symbolic forms: parameters and case labels. The case labels can express ordinary predicates and relations easily. Symmetric relations may be expressed by using the same label twice as in {same!size:a, same!size:b}. The only operator is the conjunction. The language has no disjunction or internal disjunction. As a result, the fact that each event in Fig. 2 contains a polygon on top of a *circle or rectangle* cannot be discovered.

(ii) *Rules of generalization.* The method uses the dropping condition and turning constants to variables rules.

(iii) *Computational efficiency.* On our test example, the algorithm requires 22 expression comparisons, and generates 20 candidate conjunctive generalizations of which 6 are retained. This gives a figure of 6/20 or 30% for computational efficiency. Four separate interference matches are required since the first match of E1 and E2 produces three possible conjunctive generalizations.

(iv) *Flexibility and extensibility.* Hayes-Roth has indicated (personal communication) that this method has been extended to produce disjunctive generalizations and to detect errors in data. Hayes-Roth has applied this method to various problems in the design of the speech understanding system Hearsay II. However, no facility has been developed for incorporating domain-specific knowledge into the generalization process.

Also, no facility for constructive induction has been incorporated although Hayes-Roth has developed a technique for converting a PSR to a lower-level finer-grained uniform PSR. This transformation permits the program to develop descriptions which involve a many-to-one binding of parameters.

2.2.3. *Vere: Program Thoth* [34–37]

Vere's earlier work on induction [34] was also directed at finding the MSC-generalizations of a set of input positive examples (in his work such generalizations are called *maximal conjunctive generalizations* or *maximal unifying generalizations*). Each example is represented as a conjunction of *literals.* A *literal* is a list of constants called *terms* enclosed in parentheses. For example, the objects in Fig. 1 would be described as:

E1: (circle a)(square b)(small a)(small b)(ontop a b)

E2: (circle c)(square d)(circle e)(small c)
(large d)(small e)(ontop c d)(inside e d)

Although these resemble Hayes-Roth's PSR's, they are quite different. There are no distinguished symbols. All terms (i.e. constants) are treated uniformly.

As in Hayes-Roth's work, Vere's method operates in a purely bottom-up fashion, in which the input examples are processed one at a time in order to build the set of conjunctive generalizations. The algorithm for generalizing a pair of events operates in four steps.

Step 1. The literals in each of the two events are matched in all possible ways to generate the set of matching pairs MP. Two literals match if they contain the same number of constants and share a common term in the same position. For the sample problem of Fig. 1, we have

MP = {((circle a),(circle c)),
((circle a),(circle e)),
((square b),(square d)),
((small a),(small c)),
((small a),(small e)),
((small b),(small c)),
((small b),(small e)),
((ontop a b),(ontop c d))}

Step 2. This step involves selecting all possible subsets of MP such that no single literal of one event is paired with more than one literal in another event. Each of these subsets eventually forms a new generalization of the original events.

Step 3. Each subset of matching pairs selected in Step 2 is extended by adding to the subset additional pairs of literals which did not previously match. A new pair ***p*** is added to a subset S of MP if each literal in ***p*** is related to some other pair ***q*** in S by a common constant in a common position. For example, if S contained the pair ((square b), (square d)), then we could add to S the pair ((ontop a b),(inside e d)) because the third element of (ontop a b) is the second element of (square b), and the third element of (inside e d) is the second element of (square d) (Vere calls this a 3-2 relationship). New indirectly related pairs are merged into S until no more can be added.

Step 4. The resulting set of pairs is converted into a new conjunction of literals by merging each pair to form a single literal. Terms which do not match are turned into new terms which may be viewed formally as variables. For example, ((circle a),(circle c)) would be converted to (circle v1).

Evaluation. (i) *Representational adequacy.* When applied to the test example (Fig. 2), this algorithm produces many generalizations. A few of the significant ones are listed below.

1. (ontop v1 v2)(medium v1)(large v2)(blank v2)(blank v3)(shaded v4) (v5 v4)

There is a medium object on top of a large blank object. Another object is blank. There is a shaded object. (The literal (v5 v4) is *vacuous* since it contains only variables. Variable v5 was derived by unifying circle and triangle.)

2. (ontop v1 v2)(blank v1)(medium v1)(v9 v1)(v5 v3 v4)(shaded v3) (v7 v3)(v6 v3)(blank v4)(large v4)(v8 v4)

There is a medium, blank object on top of some other object and there are two objects related in some way (v5) such that one is shaded and the other is large and blank.

3. (ontop v1 v2)(medium v1)(blank v2)(large v2)(v5 v2)(shaded v3) (v7 v3)(blank v4)(v6 v4)

There is a medium object on top of a large blank object. There is a shaded object and there is a blank object.

The representation is basically a LISP-style list structure, and consequently has very little logical structure. By convention the first symbol of a literal can be interpreted as a predicate symbol. The algorithm, however, treats all terms uniformly. This relaxation of semantic constraints creates difficulties. One difficulty is that the algorithm generates vacuous literals in certain situations.

For instance, Step 3 of the algorithm allows (circle a) to be paired with (triangle b) to produce the vacuous literal (v5 v4) as in generalization 1 above. Although these vacuous literals could easily be removed after being generated, the algorithm would perform more efficiently if it did not generate them in the first place. A second difficulty resulting from the relaxation of semantic constraints is that the algorithm creates generalizations involving a many-to-one binding of variables. While such generalizations may be desirable in some situations, their uncontrolled generation is computationally expensive.

The description language contains only the conjunction operator. No disjunction or internal disjunction is included.

(ii) *Rules of generalization.* The algorithm implements the dropping condition rule and the turning constants to variable rule.

(iii) *Computational efficiency.* From the published articles [34–37] it is not clear how to perform Step 2. The space of possibilities is very large and an exhaustive search could not possibly give the computation times which Vere has published. It would be interesting to find out what heuristics are being used to guide the search.

(iv) *Flexibility and extensibility.* Vere has published algorithms which discover descriptions with disjunctions [36] and exceptions (which he calls *counterfactuals* [37]). He has also developed techniques to generalize relational production rules [35, 36]. The method has been demonstrated using the traditional AI toy problems of IQ analogy tests and blocks-world sequences. A facility for using background information to assist the induction process has also been developed. It uses a spreading activation technique to extract relevant relations from a knowledge base and add them to the input examples prior to generalizing them. The method has been extended to discover disjunctions and exceptions. It is not clear how well the method would work in noisy environments.

2.3. Model-driven methods: Buchanan et al., and Michalski

2.3.1. *Buchanan et al.*: *Program Meta-*DENDRAL [2–4, 32]

The algorithm described here is taken from the RULEGEN program (part of the Meta-DENDRAL system). Meta-DENDRAL was designed to discover cleavage rules to explain mass spectrometry data. In this application, no single cleavage rule (or equivalently, no conjunctive generalization) can be expected to explain all of the data in a particular experiment. Consequently, the program does not search for MSC-generalizations. Instead, it develops a taxonomic description of the mass spectrometry data in the form of a *set* of cleavage rules which together cover all examples. Thus Meta-DENDRAL can be considered to perform a kind of conceptual clustering. Because the goal description sought by Meta-DENDRAL is not a conjunctive generalization, it is difficult to compare this method with the previous methods presented in this paper.

The description language used in Meta-DENDRAL is based on the ball-and-stick model of chemical molecules. Each input event is a very specific cleavage rule which predicts that a particular bond in a particular part of a molecule will be cleaved (broken) when that molecule is placed in a mass spectrometer. Each cleavage rule is thus a condition-action rule. The condition part of the rule is a bond environment which describes some portion of a molecule. The environment is represented as a graph of the atoms in the molecule with four descriptors attached to each atom. The action part of the rule indicates (by **) the bond which is predicted to break. A typical cleavage rule (with atoms w, x, y, and z) is:

CONDITION PART (BOND ENVIRONMENT):

Molecule graph: w -- x -- y -- z --

Atom descriptors:

atom	type	nhs	nbrs	dots
w	carbon	3	1	0
x	carbon	2	2	0
y	nitrogen	1	2	0
z	carbon	2	2	0

ACTION PART (CLEAVAGE PREDICTION):

w ** x -- y -- z --

The atom descriptors have the following meanings. *Type* is the atomic element of the atom. *Nhs* is the number of hydrogen atoms bound to that atom. *Nbrs* is the number of non-hydrogen atoms bound to the atom. *Dots* counts the number of unsaturated valence electrons of the atom. This rule says that whenever a molecule containing the four atoms w, x, y, and z (connected as shown in the molecule graph and with the indicated atom descriptors) is placed in a mass spectrometer, then the bond joining w to x will be broken.

RULEGEN is given a large set of these specific cleavage rules (developed by the program INTSUM from observations of the behavior of example molecules in the mass spectrometer). The goal of the program is to produce a small set of generalized cleavage rules which cover most of the input rules. The algorithm chooses as its starting point the most general cleavage prediction (x ** y) with no properties specified for either atom. During the search, this description is grown by successively specializing a property of one of the atoms in the graph, or by adding a new atom to the graph. After each specialization, the new graph is checked to see if it is 'better' than the parent graph from which it was derived. A 'child' graph is better than its 'parent' if it still explains peaks in the spectra of at least half of the molecules under consideration (it's general enough) and focuses on roughly one cleavage process per molecule (it's specific enough). The cleavage rules built by this algorithm are further improved by the program RULEMOD.

Evaluation. (i) *Representational adequacy.* The representation was adequate for the task of developing cleavage rules. It was specifically designed for use in chemical domains and is not general. The descriptions can be viewed as conjunctions. Individual rules developed by the program can be considered to be linked by disjunction.

(ii) *Rules of generalization.* The dropping condition and turning constants to variables rules are used 'in reverse' during the specialization process. Meta-DENDRAL also uses the generalization by internal disjunction rule. For example, it can learn that the type of an atom is 'anything except hydrogen'. In related work on nuclear magnetic resonance (NMR), Schwenzer and Mitchell [32] present an example in which the value of *nhs* is listed as 'greater than or equal to one' (which indicates an internal disjunction).

(iii) *Computational efficiency.* The comparison of computational efficiency is not provided for Meta-DENDRAL, because it is not possible to hand-simulate its operation on the sample problem of Fig. 2. First of all, it is impossible to represent the sample problem as a chemical graph because the problem uses *two* different connecting relationships (*ontop* and *inside*) whereas Meta-DENDRAL only allows one (chemical bonding). Secondly, as mentioned above, the algorithm seeks a taxonomic—not characteristic—description of the input examples. Thirdly, the termination criteria for the RULEGEN algorithm are stated in purely chemical terms which have no counterpart in the domain of geometric figures. The current program is considered to be relatively inefficient [3].

(iv) *Flexibility and extensibility.* Meta-DENDRAL has been extended to handle NMR spectra. The program works well in an errorful environment. It uses domain-specific knowledge extensively. However, there is no strict separation between a general-purpose induction component and a special-purpose knowledge component. It is not clear whether the methods developed for Meta-DENDRAL could be easily applied to any non-chemical domain.

Meta-DENDRAL has extensive constructive induction facilities. In particular, program INTSUM performs sophisticated transformations of the input spectrum in order to develop the bond-environment descriptions. Unfortunately, this part of the program is highly procedural. None of the rules of constructive induction have been made explicit, nor is there a general facility for accepting additional rules of constructive induction from the user.

2.3.2. *Michalski and Dietterich: Program* INDUCE 1.2

Michalski and his collaborators have worked on many aspects of induction. Most relevant here are works by Larson and Michalski [15, 16, 20], which describe a general method (and program) for determining disjunctive structural descriptions which can also be used (somewhat inefficiently) to discover MSC-generalizations. The method presented here is specially designed for finding MSC-generalizations.

The language used to describe the input events is VL_{21} [20]: an extension to first-order predicate logic (FOPL) which was developed specifically for use in inductive inference. Each event is represented as a conjunction of *selectors*. A *selector* is a relational statement which typically contains a function or predicate descriptor (with variables as arguments) and a list of values which the descriptor may assume. For example, the selector [size(v1) = small, medium] asserts that the size of v1 may take the values small or medium. Another form of selector is an n-ary predicate in brackets, which is interpreted in the same way as in FOPL. For example, the selector [ontop(v1, v2)] asserts that object v1 is ontop of object v2. A conjunction of selectors is denoted by their concatenation. The events in Fig. 1 are represented as

E1: ∃v1, v2 [size(v1) = small][size(v2) = small]
[shape(v1) = circle][shape(v2) = square][ontop(v1, v2)]

E2: ∃v1, v3 [size(v1) = small][size(v2) = large]
[size(v3) = small][shape(v1) = circle]
[shape(v2) = square][shape(v3) = circle]
[ontop(v1, v2][inside(v3, v2)]

In this method, we attempt to accelerate the search for plausible generalizations by using techniques similar to those of hierarchical planning [31]. First, we separate all descriptors into two classes: unary and non-unary descriptors. We call the unary descriptors *attribute descriptors* since they are typically used to represent attributes such as size or shape. Non-unary descriptors are called *structure-specifying descriptors* since they are typically used to specify structural information (for example relationships ontop and inside).

The basic idea of the method is to first search the description space which is defined by the structure-specifying descriptors. Once plausible generalizations are found in the abstract *structure-only space*, attribute descriptor space is searched to fill out the detailed generalizations. There are several advantages to this two phase approach.

The first is representational. As we have seen above, it is usually necessary to use a graph (or equivalent data structure) to represent an event in a structural learning problem. This is due to the fact that a graph is the most compact way to represent binary relationships among n objects when the number of such relationships is substantially less than the $n(n-1)$ possible relationships (i.e. when the relationship matrix is sparse). Thus, in our method, the structure-only events are represented as graphs. But once we have located plausible points in this structure-only space, we can continue the search in attribute space. Attribute (or unary) descriptors can be represented as vectors which are substantially more compact and more efficiently manipulated than graphs.

The second advantage of this hierarchical approach is computational. The task of comparing two graph structures is NP-complete. Any decrease in the size of these graph structures leads to large decreases in the cost of a graph comparison. Furthermore, we can confine graph comparisons to the first phase of the algorithm.

A third advantage of this approach is that we can take 'large steps' during the search for plausible descriptions by conducting much of the search in a sparse, abstract space.

There are also several disadvantages to this approach. Firstly, no speedup will be obtained unless the learning problem uses both unary and non-unary descriptors. There are some learning problems in which attributes play almost no role at all. In such cases, the structure-only search space is the same as the complete search space, so no computational savings will be obtained. There are also learning problems which require only unary descriptors (as in [12]). These are not structural learning problems, and the structure-only space is empty.

A second disadvantage of this approach involves the problem of defining 'plausible' descriptions in structure-only space. One fact which can be used is the following: If g is a MSC-generalization in structure-only space, then there exists a full description $\mathbf{G}$, such that g is the structure-only portion of $\mathbf{G}$, and $\mathbf{G}$ is a MSC-generalization in the complete space.

Thus, if we find all MSC-generalizations of the input events in structure-only space, then we can use these to find MSC-generalizations in the complete space. However, we will not necessarily find all possible MSC-generalizations in this fashion, since there may exist MSC-generalizations in the complete space whose structure-only component is *not* maximally specific in structure-only space. To avoid this problem, the algorithm may accept less than maximally specific generalizations in the structure-only space (i.e. more general descriptions), and terminate the search using some problem-oriented knowledge.

Another difficulty concerns how to conduct the attribute search once plausible structure-only descriptions have been located. Our approach is to use each structure-only description to define a new attribute-only space into which all of the input events are translated. Unfortunately, an input event can be mapped to more than one attribute-only description as shown below. This complicates the search.

The algorithm searches structure-only space using a 'beam search'—a form of best-first search in which a *set* of best candidate descriptions is maintained during the search (see [30]). First, all unary descriptors are removed from the input events (thus abstracting them into structure-only space). Then a random sample of these events is taken to form the set B_0, the initial set of generalizations (the initial beam set). In each step, B_i is first pruned to a fixed sized *beam width* by removing unpromising generalizations. (Promise is determined by the application of the heuristic evaluation functions described below.) Then B_i is checked to see if any of its generalizations covers all of the input examples. If any do, they are removed from B_i and placed in the set C of candidate conjunctive generalizations. Lastly, B_i is generalized to form B_{i+1} by taking each element of B_i and generalizing it in all possible ways by dropping single selectors. When the set of candidates C reaches a prespecified size, the search halts. The set C contains conjunctive generalizations of the input data, some of which are maximally specific. The size limit on C determines how deeply the algorithm searches.

The program allows the user to employ simultaneously several criteria for evaluating the promise of intermediate generalizations. These criteria are combined to form a lexicographic evaluation functional with tolerances [17]. Some of the criteria presently included in the program are

(a) maximize the number of input events covered by a generalization,

(b) maximize the number of selectors in a generalization, and

(c) minimize the total 'cost' of the descriptors in a generalization. Different descriptors can be given costs according to their difficulty of measurement and other domain-dependent properties.

The user creates the evaluation functional by selecting crtieria from a list of available criteria and ordering them in decreasing order of importance. Each criterion is accompanied by a tolerance which specifies the allowed departure of the associated criterion from the optimum value (see [17]).

Once the structure-only candidate set C has been built, each candidate generalization in C must be filled out by finding values for its attribute descriptors. Each candidate generalization $\boldsymbol{g}$ in C is used to define an attribute-only space which is then searched using a beam search technique similar to that used to search the structure-only space. The attribute-only space is defined as follows. Let $\{\nu_1, \nu_2, \dots, \nu_k\}$ be the existentially quantified variables used in the candidate structure-only generalization $\boldsymbol{g}$. The attribute-only space generated by $\boldsymbol{g}$ is the space of all $m \times k$-tuples consisting of the values of the m attributes describing the k objects denoted by the quantified variables $\{\nu_1, \nu_2, \dots, \nu_k\}$. In cases where some of the m attributes are not applicable to some of the objects, the attribute-only space will be correspondingly smaller.

In order to search this space, all of the input events must first be translated into this attribute-only space. This is accomplished by matching $\boldsymbol{g}$ against all input events, and extracting the attributes of the variables in the input events which match $\nu_1, \nu_2, \dots, \nu_k$ in $\boldsymbol{g}$. The values of these attributes form a single $m \times k$-tuple. For example, if $\boldsymbol{g} = $ [ontop(v1, v2)] and the variables v1 and v2 have two attributes, size and shape, then the attribute-only space generated by $\boldsymbol{g}$ is the space of all 4-tuples of the form

$$\langle \text{size}(v1), \text{size}(v2), \text{shape}(v1), \text{shape}(v2)\rangle$$

Let E_1 be the following input event:

$$\begin{aligned}E_1\colon\ & \exists p1, p2, p3\,[\text{ontop}(p1, p2)][\text{ontop}(p2, p3)]\\ & [\text{size}(p1) = 1][\text{size}(p2) = 3][\text{size}(p3) = 5]\\ & [\text{color}(p1) = \text{red}][\text{color}(p2) = \text{green}][\text{color}(p3) = \text{blue}]\end{aligned}$$

Then we can translate E_1 into this attribute-only space in two different ways—since g matches E_1 in two distinct ways. When g is matched to E_1 so that v1 is matched with p1, and v2 with p2, the resulting attribute-only 4-tuple is

$$\langle 1, 3, \text{red}, \text{green}\rangle$$

When v1 is matched to p2, and v2 to p3, then the resulting event is

$$\langle 3, 5, \text{green}, \text{blue}\rangle$$

During the search of this attribute-only space, the goal is to find an MSC-generalization which covers at least one of these two translated events (and thus covers E_1). Such an MSC-generalization is in the form of an $m \times k$-tuple as above, except that each position in the tuple may contain a *set* of values of the corresponding attribute. This set of values is expressed by an internal disjunction in the final corresponding formula.

The beam search of attribute-only space is similar to the search of structure-only space. A random sample of events is selected and generalized step-by-step by extending the internal disjunctions in the events. The generalization process is guided by a means-ends analysis to detect relevant differences between the current generalizations and events which have not yet been covered. Heuristic criteria are used to prune the beam set to a fixed beam width. Candidate generalizations which cover all of the input events (i.e. at least one of the attribute-only events translated from each input event) are removed from the beam set and added to the candidate set C'. Each candidate in C' provides possible settings of the attribute descriptors which, when combined with the structure-specifying descriptors in g, produces an output conjunctive generalization **G**.

Among all conjunctive generalizations produced by this algorithm, there may be some which are not maximally specific. This occurs when the search of structure-only space is permitted to produce candidate structure-only generalizations which are not maximally specific. In most observed cases such candidate generalizations become maximally specific when their attribute descriptors are filled in during the second phase of the algorithm.

Evaluation. (i) *Representational adequacy.* Using only non-constructive rules of generalization, the algorithm discovers, among others, the following generalizations of the events in Fig. 2.

1. $\exists$v1, v2 [ontop(v1, v2)][size(v1) = medium][shape(v1) = polygon]
[texture(v1) = blank][size(v2) = medium, large]
[shape(v2) = rectangle, circle]

There exist two objects (in each event), such that one is a blank, medium-sized polygon on top of the other, a medium or large circle or rectangle.

2. ∃v1, v2 [ontop(v1, v2)][size(v1) = medium]
[shape(v1) = circle, square, rectangle][size(v2) = large]
[shape(v2) = box, rectangle, ellipse][texture(v2) = blank]

There exist two objects such that one of them is a medium-sized circle, rectangle, or square on top of the other, a large, blank box, rectangle, or ellipse.

3. ∃v1, v2 [ontop(v1, v2)][size(v1) = medium][shape(v1) = polygon]
[size(v2) = medium, large][shape(v2) = rectangle, ellipse, circle]

There exist two objects such that one of them is a medium-sized polygon on top of the other, a large or medium rectangle, ellipse, or circle.

4. ∃v1 [size(v1) = small, medium][shape(v1) = circle, rectangle]
[texture(v1) = shaded]

There exists one object, a medium or small shaded circle or rectangle.

A few simple constructive induction rules have been incorporated into the current implementation. These include rules which count the number of objects possessing certain characteristics, and rules which locate the top-most and bottom-most parts of an object (or more generally, extremal elements in a linearly-ordered set defined by any transitive relation, e.g., *ontop*). Other constructive induction rules can be specified by the user. Using the built-in constructive induction rules, the program produces the following conjunctive generalization of the input events in Fig. 2.

5. [# v's = 3, 4][# v's with texture blank = 2] ∧
∃v1, v2 [top-most(v1)][ontop(v1, v2)][size(v1) = medium]
[shape(v1) = polygon][texture(v1) = clear]
[size(v2) = medium, large][shape(v2) = circle, rectangle]

There are either three or four objects in each event. Exactly two of these objects are blank. The top-most object is a medium sized, clear polygon and it is on top of a large or medium sized circle or rectangle.

This algorithm implements the conjunction, disjunction, and internal disjunction operators. The representation distinguishes among descriptors, variables, and values. Descriptors are further divided into structure-specifying descriptors and attribute descriptors. The current method discriminates among three types of descriptors:
- nominal—which have unordered value sets,
- linear—which have linearly-ordered value sets, and
- structured—which have tree-ordered value sets.

This variety of possible representational forms is intended to provide a better 'fit' between the description language and any specific problem.

TABLE 2. Comparison of different learning methods

Method Criterion:	Winston	Hayes-Roth	Vere	Buchanan et al.	Michalski
Intended application	Learning toy block concepts	General	General	Discovering mass spectrometry rules	General
Language	Semantic net	Parameterized structural representation	Quantifier-free FOPL	Chemical model	Variable-valued logic system VL_{21}
Syntactic concepts	Nodes and links of many types	Case frames parameters case labels	Literals constants	Molecule graph attributes constants in value sets	Selectors descriptors variables value sets
Operators	$\wedge$, exception	$\wedge$	$\wedge$	$\wedge$, $\vee$, internal $\vee$	$\wedge$, $\vee$, internal $\vee$
Generalization and specialization rules:					
Dropping condition?	Yes	Yes	Yes	Yes	Yes
Constants to variables?	Yes	Yes	Yes	Yes	Yes
Generalizing by internal $\vee$?	No	No	No	Yes	Yes
Climbing tree?	Yes	No	No	No	Yes
Closing intervals?	No	No	No	No	Yes
Introducing exceptions?	Yes	No	No	No	No
Efficiency:					
Graph comparisons	2	22	Complete algorithm not known	Not applicable	28
Graph generations during search	11	20		Not applicable	13
Ratio of output to total	1/11 = 9%	6/20 = 30%		Not applicable	4/13 = 30%
Extensibility:					
Applications	None	Speech analysis	None	Mass spectrometry, NMR	Soybean disease diagnosis
Disjunctive forms?	No	No	Yes	Yes	Yes
Noise immunity?	Very low	Low	Probably good	Excellent	Very good
Domain knowledge?	Yes, built-in to program	No	Yes	Yes, built-in to program	Yes
Constructive induction?	Limited facility	No	No	Extensive problem-specific facility	Few general rules

(ii) *Rules of generalization.* The algorithm uses all rules of generalization mentioned in Section 1.5, and also a few constructive induction rules. It does not implement the introducing exception specialization rule. The effect of the turning constants to variables rule is achieved as a special case of the generalization by internal disjunction rule.

(iii) *Computational efficiency.* The algorithm requires 28 comparisons, and builds 13 rules during the search to develop the descriptions listed above. Four rules are retained, so this gives an efficiency ratio of 4/13 or 30%.

(iv) *Flexibility and extensibility.* The algorithm can be modified to discover disjunctions by altering the termination criteria for the search of structure-only space to accept structure conjuncts which do not necessarily cover all of the input events. The same general two-phase approach can also be applied to problems of determining discriminant generalizations. Larson and Michalski [15–18, 20] have done work on determining such discriminant descriptions.

The algorithm has good noise immunity. Noise events can be discovered because the algorithm tends to place them in separate terms of a disjunction.

Domain-specific knowledge can be incorporated into the program by defining the types and domains of descriptors, specifying the structures of these domains, specifying certain simple production rules (for domain constraints on legal combinations of variables), specifying the evaluation functional, and by providing constructive induction rules. These forms of knowledge representation are not always convenient, however. Further work should provide other facilities for knowledge representation.

As mentioned above, the method does perform a few general kinds of constructive induction. The method provides mechanisms for adding more rules of constructive induction.

The comparison of the above methods in terms of the criteria of Section 2.1 is summarized in Table 2.

3. Summary and Suggested Topics for Further Research

We have discussed various aspects of structural machine learning and introduced several criteria for evaluating learning methods. These criteria have been applied to the evaluation of five selected methods of learning maximally specific conjunctive descriptions, including the author's own method. One of the features revealed by the analysis is that top-down and bottom-up methods present a tradeoff between computational efficiency on the one hand, and flexibility and extensibility on the other. Bottom-up methods tend to be faster, but have lower noise immunity and less flexibility. Top-down methods have good noise immunity and can be easily modified to discover disjunctive and other forms of generalization. They do tend to be computationally more expensive.

Another important point brought out by the analysis is the importance of selecting an appropriate description language for research in inductive learning. A learning method which uses a language with little structure (i.e., which has few operators and few types of operands) tends to be relatively efficient and easy to implement, but may not be able to learn descriptions which are most useful in real-world applications. On the other hand, a method which uses a language which is too rich will lead to enormous implementation problems which will be detrimental to successful research in machine learning.

A significant problem in current research on inductive learning is that each research group is using a different notation and terminology. This not only makes the exchange of research results difficult, but it also makes it hard for new researchers to enter the field. This paper has attempted to develop a set of concepts and criteria which abstract from these differences in notation and terminology.

The analysis raises some important problems to be addressed in future research:

(i) *Further work on representations.* Present learning programs are limited by the kinds of operators and variable types they allow, and also by the forms of descriptions they can produce. Methods for handling additional operators, variable types, and forms of descriptions need to be designed and implemented. Rules of generalization corresponding to these operators, types, and forms should also be developed. Among forms which are particularly desirable are hierarchical and related forms in which a name of one description is used to build other, more complex descriptions. Some initial work in this area has been done by Winston [38, 39], Cohen and Sammut [5], and in the area of grammatical inference in general (see Biermann [1]).

(ii) *The Principle of Comprehensibility.* In applications where people will need to use the generalizations produced by a learning program, it is important that the learning method produces generalizations which are easy to understand and close to corresponding natural language descriptions. This means that the descriptions developed by an inductive method must be structured to take into consideration human information processing limitations. As a rough guideline, conjunctions should involve no more than three or four conditions, full descriptions should involve only two or three disjunctive terms, and there should be no more than two quantifiers in the description. Descriptions should correspond to single 'chunks' of information. Hierarchically structured descriptions may provide a way to meet these guidelines.

(iii) *Constructive induction.* The constructive induction techniques developed to date are very limited. New rules of constructive induction need to be identified and implemented. An important problem is the development of efficient mechanisms for guiding the process of constructive induction through the potentially immense space of possible derived descriptors.

(iv) *Integration of problem-specific knowledge.* Further work should be done on the problem of when and how to use problem-specific knowledge in a general induction method. The use of typed variables is a good example of a general way to incorporate problem-specific knowledge.

(v) *Extension to discriminant and taxonomic descriptions.* Much work has been done on characteristic generalization. Discriminant and taxonomic descriptions are very important, especially in noisy environments. More work on this subject is needed.

(vi) *User interface.* As AI learning programs become more powerful, their functions will become more opaque. Learning programs should provide explanation facilities for justifying their generalizations.

(vii) *Handling errors and missing data.* Very little attention has been paid to the problem of developing methods which work well in noisy environments. There is need for research on methods of learning from uncertain input information, from incomplete information, and from information containing errors.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support of NSF under grants MCS-76-22940 and MCS-79-006614. The authors thank the reviewers for their thorough review and criticism which helped significantly to clarify some of the concepts and to improve the organization of the final version of this paper.

REFERENCES

1. Biermann, A. and Feldman, J., A survey of results in grammatical inference, in: Watanabe, S. (Ed.), *Frontiers in Pattern Recognition* (Academic Press, New York, 1972).
2. Buchanan, B.G., Feigenbaum, E.A. and Lederberg, J., A heuristic programming study of theory formation in science, *Proc. of the 2nd Internat. Joint Conf. on Artificial Intelligence* (1971) 40–48.
3. Buchanan, B.G., Smith, D.H., White, W.C., Gritter, R.J., Feigenbaum, E.A., Lederberg, J. and Djerassi, C., Applications of artificial intelligence for chemical inference, XXII. Automatic rule formation in mass spectrometry by means of the Meta-DENDRAL program, *Journal of the American Chemical Society* **98** (1976) 6168.
4. Buchanan, B.G. and Feigenbaum, E.A., Dendral and Meta-DENDRAL, their applications dimension, *Artificial Intelligence* **11** (1978) 5–24.
5. Cohen, B. and Sammut, C., Pattern recognition and learning with a structural description language, *Proceedings of the Fourth Internat. Joint Conf. on Pattern Recognition*, Kyoto, 1978.
6. Dietterich, T.G., User's guide for INDUCE 1.1, Internal Rep., Department of Computer Science, University of Illinois, Urbana (1978).
7. Fikes, R.E., Hart, P.E. and Nilsson, N.J., Learning and executing generalized robot plans, *Artificial Intelligence* **3** (1972) 251–288.
8. Hayes-Roth, F., Collected papers on learning and recognition of structured patterns, Department of Computer Science, Carnegie–Mellon University, Pittsburgh (January 1975).
9. Hayes-Roth, F., Patterns of induction and associated knowledge acquisition algorithms, Department of Computer Science, Carnegie–Mellon University, Pittsburgh (May 1976).
10. Hayes-Roth, F. and McDermott, J., Knowledge acquisition from structural descriptions, *Proceedings of the 5th Internat. Joint Conf. on Artificial Intelligence* (1977) 356–362.
11. Hayes-Roth, F. and McDermott, J., An interference matching technique for inducing abstractions, *Comm. ACM.* **21**(5) (1978) 401–410.

12. Hunt, E.B., Marin, J. and Stone, P.J., *Experiments in Induction* (Academic Press, New York, 1966).
13. Knapman, J., A critical review of Winston's learning structural descriptions from examples, *AISB Quarterly* **31** (1978) 319–320.
14. Lenat, D., AM: An artificial intelligence approach to discovery in mathematics as heuristic search, Internal Rept. STAN-CS-76-570, Computer Science Department, Stanford University, Stanford (July 1976).
15. Larson, J. and Michalski, R.S., Inductive inference of VL decision rules, *SIGART Newsletter* (June 1977) 38–44.
16. Larson, J., Inductive inference in the variable valued predicate logic system VL_{21}: methodology and computer implementation, Internal Rept. No. 869, Department of Computer Science, University of Illinois, Urbana (1977).
17. Michalski, R.S., Discovering classification rules using variable-valued logic system VL_1, *Proceedings of the 3rd Internat. Joint Conf. on Artificial Intelligence* (1973) 162–172.
18. Michalski, R.S., Variable-valued logic and its application to pattern recognition and machine learning, in: Rine, D.C. (Ed.), *Computer Science and Multiple-Valued Logic* (North-Holland, Amsterdam, 1977) 506–534.
19. Michalski, R.S., Toward computer-aided induction: a brief review of currently implemented AQVAL programs, *Proceedings of the 5th Internat. Joint Conf. on Artificial Intelligence* (1977) 319–320.
20. Michalski, R.S., Pattern recognition as rule-guided inductive inference, *IEEE Trans. Pattern Analysis and Machine Learning* **2**(4) (1980) 349–361.
21. Michalski, R.S., Knowledge acquisition through conceptual clustering: a theoretical framework and an algorithm for partitioning data into conjunctive concepts, *Internat. J. Policy Analysis and Information Systems* **4**(3) (1980).
22. Michalski, R.S., Inductive learning as rule-guided generalization and conceptual simplification of symbolic descriptions: unifying principles and a methodology, *Papers of the Workshop on Machine Learning*, Carnegie–Mellon University, Pittsburgh (July 1980).
23. Michie, D., New face of AI, Experimental Programming Rept. No. 33, Machine Intelligence Research Unit, University of Edinburgh, Edinburgh (1977).
24. Mitchell, T.M., Version spaces: a candidate elimination approach to rule learning, *Proceedings of the 5th Internat. Joint Conf. on Artificial Intelligence* (1977) 305–310.
25. Mitchell, T.M., Version spaces: an approach to concept learning, Internal Rept. STAN-CS-78-711, Computer Science Deparment, Stanford University, Stanford (1978).
26. Plotkin, G.D., A note on inductive generalization, in: Meltzer, B. and Michie, D. (Eds.), *Machine Intelligence* **5** (Elsevier, New York, 1970).
27. Plotkin, G.D., A further note on inductive generalization, in: Meltzer, B. and Michie, D. (Eds.), *Machine Intelligence* **6** (Edinburgh University Press, Edinburgh, 1971).
28. Quinlan, J.R., Induction over large data bases, Internal Rept. HPP-79-14, Heuristic Programming Project, Stanford University, Stanford (1979).
29. Quinlan, J.R., Discovering rules from large collections of examples: a case study, in: Michie, D. (Ed.), *Expert Systems in the Micro Electronic Age* (Edinburgh University Press, Edinburgh, 1979).
30. Rubin, S.M. and Reddy, R., The locus model of search and its use in image interpretation, *Proceedings of the 5th Internat. Joint Conf. on Artificial Intelligence* (1977) 590–595.
31. Sacerdoti, E., Planning in a hierarchy of abstraction spaces, *Proceedings of the 3rd Internat. Joint Conf. on Artificial Intelligence* (1973) 412–422.
32. Schwenzer, G.M. and Mitchell, T.M., Computer-assisted structure elucidation using automatically acquired carbon-13 NMR rules, in: Smith, D.H. (Ed.), *Computer-assisted Structure Elucidation*, ACS Symposium Series, No. 54 (1977).
33. Stepp, R., Learning without negative examples via variable-valued logic characterizations: the

UNICLASS induction program AQ7UNI, Internal Rept. UIUC-DCS-79-982, Department of Computer Science, University of Illinois, Urbana (1979).

34. Vere, S.A., Induction of concepts in the predicate calculus, *Proceedings of the 4th Internat. Joint Conf. on Artificial Intelligence* (1975) 281–287.
35. Vere, S.A., Induction of relational productions in the presence of background information, *Proceedings of the 5th Internat. Joint Conf. on Artificial Intelligence* (1977) 349–355.
36. Vere, S.A., Inductive learning of relational productions, in: Waterman, D.A. and Hayes-Roth, F. (Eds.), *Pattern-Directed Inference Systems* (Academic Press, New York, 1978).
37. Vere, S.A., Multilevel counterfactuals for generalization of relational concepts and productions, Department of Information Engineering, University of Illinois, Chicago Circle (1978).
38. Winston, P.H., Learning structural descriptions from examples, Ph.D. thesis, Report AI-TR-231, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge (1970).
39. Winston, P.H., Learning structural descriptions from examples, in: Winston, P.H. (Ed.), *The Psychology of Computer Vision* (McGraw-Hill, New York, 1975).

Received November 1979; revised version received November 1980