Inductive Concept Learning using the Artificial Intelligence Approach

by

Bruce F. Cockburn

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Research Report CS-85-12
May 1985

Inductive Concept Learning using the Artificial Intelligence Approach

Bruce F. Cockburn

M695: Master's Essay
Supervisor: Marlene Jones
Department of Computer Science
University of Waterloo

Copyright © 1985 by Bruce F. Cockburn

ABSTRACT

This essay is a critical survey of past attempts to build programs that learn symbolic concepts by induction from examples. Specifically, emphasis is placed on programs that use Artificial Intelligence techniques as opposed to the alternative methods of numerical and statistical analysis. First, the distinguishing characteristics of previous programs are described to provide criteria for the evaluation of actual systems. Then, learning programs developed by Winston, Vere, Buchanan, Mitchell, Quinlan, Michalski and Stepp are presented and their properties discussed. A critical discussion outlines areas of weakness in past research. Finally, appropriate directions for future research are identified.

Acknowledgements

The author would like to extend his warmest thanks to his supervisor Marlene Jones of the University of Waterloo. Marlene's guidance at our bi-weekly meetings, her patience as the first few months seemed to slip by without producing tangible results, and her willingness to read several drafts of this essay were gratefully appreciated. I would also like to acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada.

Table of Contents

Inductive Concept Learning using the Artificial Intelligence Approach

Bruce F. Cockburn

M695: Master's Essay
Supervisor: Marlene Jones
Department of Computer Science
University of Waterloo

Copyright © 1985 by Bruce F. Cockburn

1. Introduction

1.1. Why Machine Learning is Important

Machine learning is an important area of research in Artificial Intelligence (AI) principally because the ability to learn is widely considered to be a fundamental characteristic of naturally occurring intelligence. Learning has proved to be a particularly difficult behaviour to duplicate using computer programs. Today, after more that twenty years of research, the performance of learning algorithms is still disappointingly poor by human standards.

A second important reason for current interest in machine learning has arisen with the recent successes of expert systems and with an increased awareness of the potential of larger knowledge-based systems. Such computer models of human expertise are typically time-consuming to build because of the difficulty of acquiring and encoding human knowledge. For knowledge to be translated into machine usable form, a highly trained knowledge engineer must spend months and often years in close collaboration with a human expert in the desired domain. Machine learning offers the prospect of transferring much of the tedious burden of encoding human knowledge from the knowledge engineer to the target computer. The computer could then assume the responsibility of finding and using its own appropriate data structures for storing human knowledge. Once one computer has learned, many other computers can become identical experts simply by copying the learned knowledge base.

1.2. Overview

This essay describes past attempts to devise computer algorithms that exhibit what is believed to be a relatively simple form of learning: the inductive learning of symbolic concepts from examples. Learning by example has received more attention than other types of learning such as learning by being told, learning by discovery and learning by analogy. The common characteristics and parameters of the many published algorithms are described to obtain an appropriate perspective before considering six illustrative programs in more detail. A critical discussion identifies the strengths and weaknesses of existing programs and a concluding section suggests directions for future research.

1.3. Definitions of Learning

Before one can meaningfully discuss programs that claim to learn, it is important that a definition of the behaviour called learning be understood. Unfortunately, there is no concise, universally accepted definition. As a first attempt, consider the definition for the verb 'to learn' given in the Webster's New Collegiate Dictionary:

> 'to gain knowledge or understanding of or skill in by study, instruction or experience' [26]

This definition might be useful to a foreigner wishing to confirm the English translation for a word that they already know. Its vagueness makes it inadequate for the purposes of evaluating progress in machine learning. A more suitable definition is given by Simon in his skeptical introductory chapter to the book *Machine Learning: An Artificial Intelligence Approach* :

> 'learning is any change in a system that allows it to perform better the second time on repetition of the same task or on another task drawn from the same population. ... Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time.' [23]

This second definition is more useful in the context of this essay than the first, but it can be criticized for its emphasis on promptly improved, externally observable performance. It precludes the possibility that improvement might not be monotonic in the short term. Certainly humans can learn concepts without necessarily manifesting the new knowledge in an observable way. Scott prefers a definition that places more

emphasis on the internal world model that any successful learning program is assumed to maintain.

> '[learning] is any process in which a system builds a retrievable representation of its past interactions with the environment. ... Learning is the organization of experience.' [21]

A fourth definition of learning is that given by Carbonell, Michalski and Mitchell:

> 'there are two basic forms of learning: **knowledge acquisition** and **skill refinement.** Knowledge acquisition is defined as learning new symbolic information coupled with the ability to apply that information in an effective manner. ... [skill refinement] is the gradual improvement of motor and cognitive skills through practice' [5]

Programs that learn concepts from examples are expected to accomplish the knowledge acquisition part of this definition. Skill refinement is a property that usually is considered in other areas outside of AI such as the study of adaptive systems and parameter optimization [3]. In this essay, the view of learning as symbolic knowledge acquisition comes closest to a suitable working definition of learning.

1.4. The Artificial Intelligence Approach

Learning machines have been devised and even implemented in several fields (e.g. pattern recognition, adaptive control systems) but the emphasis in this essay is on learning research using the AI approach. In an AI program, symbols denoting real or abstract objects, attributes, and relations rather than numbers are manipulated during computation. Using suitable data structures as knowledge representation frameworks, the AI programmer encodes what is judged to be the relevant knowledge in the problem at hand. Constraints on the solution for the real problem being modeled must be encoded as constraints on acceptable symbolic solutions to the program. It is common to view the execution of an AI program as a search through the space of possible symbolic solutions. Typically, AI techniques are employed when there are no known efficient algorithmic solutions to a problem. In the worst cases, a solution can only be found by a brute force method like systematic search. Often the search for a solution can be made considerably more efficient if heuristics can be identified (often using insights from the real domain) that direct the search effort in the most promising directions.

1.5. Brief History

The following section recapitulates a history of machine learning given in a paper by Carbonell, Michalski and Mitchell [4]. Research on learning from examples has been influenced by the successive popularity of three learning paradigms: neural modeling and decision-theoretic methods, general symbolic concept learning, and various hybrid knowledge-intensive techniques.

In the neural modeling paradigm, popular from the late 1950's until the mid 1960's, large networks of random or partially random neuron-like switching elements were envisioned. It was expected that given some ability to re-organize themselves, such systems would be able to react to stimuli and thereby demonstrate learning behaviour. At roughly the same time, the field of pattern recognition and other related decision-theoretic theories of learning emerged. The common approach was to employ training sets to optimize the coefficients of some polynomial discriminant function. In the late 1960's and early 1970's, adaptive control systems were developed by workers in control theory. These systems learned by adjusting parameters to maintain stable performance in the presence of perturbations.

The period of popularity of the general symbolic concept learning paradigm, from the mid 1960's until the late 1970's, was the heyday for research into learning from examples. Winston's doctoral thesis of 1970 was an influential early work [27,28]. Vere's work on counterfactuals [24,25] and Mitchell's candidate learning algorithm [12,13] are just two examples of the flurry of interest that followed. A characteristic of learning programs from this period is that they learned highly structured symbolic representations of concepts embodied in a series of examples. Emphasis was placed on discovering general, domain-independent principles of learning.

Originating in the mid 1970's, the current paradigm in machine learning stresses the importance of domain-specific knowledge. Research in learning by example has continued along with increased interest in other forms of learning such as learning from instruction, learning by analogy and learning by discovery. Meta-DENDRAL, designed by Buchanan's group at Stanford University, was a successful learning program developed for the field of mass spectroscopy [1,2,3]. Quinlan produced an interesting series of programs that used human-specified features from a particular chess endgame to infer rules that could decide a game's outcome [15,16,17,18]. Michalski and Stepp worked on programs that learned taxonomic classifications from examples that corresponded well with classifications previously developed by humans [9,10,11].

2. Characteristics of Concept Learning Programs

Before considering specific algorithms in any detail, it is worthwhile to discuss the similarities and differences of all learning programs.

2.1. Representation Languages

The choice of representation languages is usually the most important decision in the design of a learning program. Languages are required to encode the input examples, the tentative and final concept descriptions, and declarative representations of heuristics. For simplicity, one all-purpose language may be preferable but separate languages for each object are sometimes used.

If the vocabulary of the language is fixed for the life of the program then the size of the space of describable concepts is also fixed. The programmer must ensure that all of the concepts that the learning program will encounter are expressible in the description language. Thus, although using a fixed language usually makes the learning algorithm simpler, it complicates the initial choice of language.

There are five major types of knowledge representation languages that have been used in learning programs [6]:

1. **The First Order Predicate Calculus:** The first order predicate calculus (FOPC) is probably the most thoroughly studied representation language. The FOPC has its origins in the study of formal logic, a field with an intellectual history extending back to the philosophers of ancient Greece. Using the full complement of operators (e.g. disjunctive, conjunctive, negative, implicative, equivalence) most objects can be described to any arbitrary level of detail. Problems occur however when objects involving probabilistic, fuzzy, default, heuristic or postulated knowledge must be represented. A significant advantage of the FOPC is that its formal semantics have been studied extensively. Resolution algorithms have been developed that, for certain subsets of logic, are guaranteed to determine whether or not a proposition is a logical consequences of known facts and rules. The programming language PROLOG is an example of the Horn clause subset of logic transformed into an executable programming language by the use of a resolution algorithm.

2. **Semantic Nets:** Semantic nets have a long history in AI research and were used in some of the earliest learning algorithms [27]. A semantic network is a set of nodes and a set of inter-nodal directed arcs. Both nodes and arcs can be labeled and assigned meanings appropriate to the particular domain. One advantage of the semantic network is the uniform character of its nodes. Hierarchies of nodes can be used to represent different levels of abstraction for related concepts or objects. Different kinds of knowledge at various levels of abstraction can be manipulated using essentially the same control mechanisms. Networks can be partitioned off to limit visibility to the relevant parts for a given context. Both an advantage and a disadvantage of semantic nets is the lack of a standardized syntax: semantic nets are hand-crafted to suit the particular needs of each problem. The semantic network formalism can be extended to have all of the logical operators found in the FOPC.

3. **Frames and Scripts:** Frames and scripts are data structures that can be useful when the common characteristics of a class of objects are known. A frame type is comprised of slots that are each capable of storing the value of one attribute for an object from the corresponding class. Default values and executable procedures can also be associated with slots: the defaults are often useful if an attribute is required but not yet available, and procedures are useful for encoding additional steps to be taken once the slot is filled with confirmed information. A script is a special type of frame that encodes a stereotypical sequence of events. Slots typically represent objects, actors and other attendant details normally encountered in the sequence. The slots in a script can also have default values. Scripts are useful because they can be used by an algorithm to anticipate or expect particular features in a situation.

4. **Production Rules:** A production rule is a pair comprised of a condition pattern and an associated action. Production rules are applied by a control mechanism with the goal of transforming a problem from its initial state to a solution state. In a typical execution cycle, the pattern portions of all known production rules are compared with the problem's current state. Those production rules whose patterns match become enabled. Using some fixed convention, or the values of some heuristic evaluation function of the states reachable from the current state, a rule is chosen and its action applied. There is no standardized syntax for production rules, but it is common to express them in a predicate logic.

5 **Hierarchical Descriptions:** A hierarchical description shows the interrelationships amongst the members of a set of objects. Closely related objects are grouped together into the same class and dissimilar objects are placed into different classes. Classes can themselves be grouped together as subclass members of a higher level class if their underlying members are sufficiently similar. Before a hierarchical description can be constructed, criteria for judging the similarity of objects and the optimal subdivision of classes into subclasses must be provided.

Scott has argued that researchers have not given enough attention to finding appropriate **knowledge** structures as opposed to the AI **data** structures discussed above [21,22]. In his opinion, a good knowledge structure must not only be sufficient to encode the desired information and be efficient for programs to manipulate, it must also be easy to augment with newly learned knowledge. According to Scott, learning researchers are confusing their data structures with the currently unknown knowledge structures that they should be using.

> 'Some authors (e.g. [5]) have suggested what appear to be classification schemes of this nature [i.e. for learned knowledge] but on closer examination they appear to be based not on the knowledge structure but rather the data structures used to represent the knowledge.' [21]

Scott raises a valid point, but does not give any examples of what he would consider to be a true knowledge structure. The search for easily learnable knowledge structures is an important area for future research.

2.2. Types of Concepts

Four main types of concepts appear in learning by example algorithms: characteristic, discriminant, taxonomic and production. The first three types are described by Dietterich and Michalski [6]. It is not claimed that these four concept types are used in their pure form by humans or that all human concepts are expressible in only these four types. They have been chosen merely as useful simplifications and abstractions of human concept structures that might serve as a basis for research in machine learning.

1. **Characteristic:** A concept is described in terms of the attributes that distinguish examples that are instances of the concept (i.e. positive examples) from examples that are not (i.e. negative examples). No knowledge is required of any other concepts that might exist in the universe: only membership with respect to the given

concept is relevant. For example, a hand mirror could be defined roughly as a flat object with a handle and one reflecting face.

2. **Discriminant:** A finite number of possible concepts are described in terms of the attributes that are sufficient to classify examples as members of one of the concepts. For example, in the domain of a particular chess endgame, each board configuration can in theory be classified as a win for white, a win for black or a stalemate, assuming that both sides make their best possible moves. It is possible to construct a decision tree which would classify all possible configurations according to their outcome. This tree would be a discriminant concept description.

3. **Taxonomic:** The desired concept is a hierarchical description or taxonomy of all the known examples. Such a description groups similar objects together and shows the probable relationships among group members. Criteria must be provided that measure the optimality of alternate classifications. An example of a taxonomic concept would be a complete classification of the different types of car body styles available in current North American automobiles.

4. **Production:** A production concept has the form of one or more production rules. A good example of a production concept learning program is Meta-DENDRAL in the domain of mass spectroscopy [1,2,3]. In this type of chemical analysis, molecules of a compound are broken up into smaller pieces and then separated out according to mass. The population of fragments has a distribution that is characteristic for each compound. Meta-DENDRAL represents molecules as nodes and arcs representing atoms and chemical bonds respectively. The fragmentation process is modeled by rules that break apart the molecules at certain bond sites. When presented with an experimentally obtained mass spectrogram for a known compound the program infers a set of production rules sufficient to account for the distribution's shape.

2.3. Resource Restrictions

Time and space resource requirements affect the usefulness of a learning algorithm. A major reason for interest in machine learning is the hope that eventually machines with a learning capability will be able to acquire knowledge faster, cheaper and more reliably than is possible by direct programming by humans. Unfortunately, most existing learning algorithms are far too slow, fragile and inflexible to be of any more than academic interest.

The human brain has an enormous amount of storage capacity and parallel computing power. On the other hand, the volume of information that is sensed from the environment over the course of years of living is huge. Much incoming information may be lost immediately. It is not clear to what extent retained information is encoded and generalized with existing knowledge, or stored as distinct memories.

Learning algorithms must also operate with finite storage. For instance, it is not advisable for designers to assume that an arbitrarily large number of examples can be saved. Many systems incorporate all examples into the current concept description. They are not saved as separate entities. Other programs incorporate only the positive examples and save the negative examples separately (it is not always clear how many can be saved). In Mitchell's version space approach, the positive and negative examples are used to update separate most specific and most general concept descriptions respectively [12,13].

2.4. Example Presentation Mode

Learning algorithms differ in how they are presented with the training examples. The entire available supply may be accessible at once, or examples may only be available one at a time. If presented incrementally, examples are available either as an uncontrollable stream or selected as requested by the learning algorithm. Control over the selection of examples allows the program to actively experiment with the environment in the ways that it decides will be most profitable. Winston's algorithm assumes that the input examples will be presented by an 'intelligent' teacher who chooses them to optimize the rate of convergence on the concept [27]. This represents the extreme case in which the environment is actively controlled to suit the learner.

2.5. Control Strategies

The best concept description can be viewed as the description that accounts for the most input data and that is optimal according to some predetermined preferences in the concept description space. It is instructive to consider the relative importance of data and predetermined biases in a learning algorithm's choice of concept description.

1. **Data-driven / bottom up:** A data-driven control strategy is one that produces concept descriptions whose form is strongly dependent on the input examples and only weakly affected by biases. The best concept evolves from the bottom up (i.e. from the data to consistent concept descriptions). An advantage of a data-driven

strategy is that the control mechanism can be relatively simple: the best tentative concept description is updated in response to each additional example. A disadvantage is that anomalous or just plain erroneous examples can disrupt or completely derail the evolving concept description.

2. **Model-driven / top down:** A model-driven control strategy is one that has strong preferences regarding the best forms of concept descriptions and that only modifies its preferred concepts enough to accommodate the data to an acceptable extent. Hypothesized concept descriptions evolve from the top down (i.e. from preferred concept descriptions to descriptions consistent with the data). Criteria must be provided that judge the adequacy of fit between data and concepts. In many domains much can be said about which forms of concept descriptions are most likely. Such information can be put to effect by a model-driven control strategy since the most likely concepts are considered before less likely candidates. On average, convergence on a good concept should be rapid. Error tolerance is higher for model-driven as opposed to data-driven algorithms because unusual examples can be more easily recognized as irregular. The disadvantages include a relatively complex control strategy and the requirement that past examples be stored to be available for checking the consistency of postulated concepts.

3. **Mixed Strategies** A compromise between the data-driven and model-driven paradigms, a mixed control strategy is one that attempts to inherit the advantages of the two previous approaches and avoid their disadvantages. For example, a mixed strategy might initially approach a learning problem in essentially a data-driven mode using weak model preferences such as favouring syntactically simple concepts. Once the character of the problem became clearer, stronger domain-specific models might then be invoked to speed up the rate of convergence. Mixed strategies occupy a poorly researched continuum between two well-defined paradigms.

2.6. Generalization Rules

A fundamental operation in any learning algorithm is generalization, expanding the scope of a concept description to include positive instances. Various rules have been identified that generalize concepts by making syntactic changes on their coded representations. Usually, there can be no guarantee that mere changes in syntax will produce useful generalizations. Good generalizations tend to be produced only when the representation language is chosen so that the syntax closely reflects the semantics

of the underlying domain [7].

The following is a list of generalization rules based on those identified by Dietterich and Michalski [6]:

1. **Drop Conditions:** A concept is generalized by dropping conditions from conjunctive expressions or subexpressions in the concept's description.

   e.g. the concept 'big(X) **and** expensive(X)'

   with the new positive example 'big(car1) **and** cheap(car1)'

   generalizes to the new concept 'big(X)'

2. **Turn Constants into Variables:** If an attribute is found that has one value in the current concept and a different value in positive examples, replace appearances of the constant with a variable that can assume any value. This generalization rule can be interpreted in different ways as Vere has pointed out [25]. A conservative interpretation changes the smallest possible number of constants into variables to match the new positive examples into the concept description. A liberal interpretation changes all occurrences of those particular constants in the concept description to variables. On empirical grounds, Vere favours a moderate interpretation that lies between these two extremes.

   e.g. the concept 'car_door_number(X,2)'

   with the new positive example 'car_door_number(car1,4)'

   generalizes to the new concept 'car_door_number(X,Y)'

3. **Add Internal Disjunction:** This is a less drastic alternative to turning constants into variables. A constant in the concept description is converted into a variable that is restricted to assuming only the values necessary to extend the concept's coverage over new positive examples. This is equivalent to introducing a new typed variable.

e.g. the concept 'shape(X,square)'

with the new positive example 'shape(part1,triangle)'

generalizes to the new concept 'shape(X,{square,triangle})'

4. **Close an Interval:** Constants belonging to an ordinal sequence are generalized to the closed interval bounded by the constants.

   e.g. the concept 'length(T,50)'

   with the new positive example 'length(trumpet1,80)'

   generalizes to the new concept 'length(T,50..80)'

5. **Climb a Generalization Hierarchy:** Domain-specific knowledge might include a hierarchy of concepts that embodies an intrinsic ordering from the specific to the more general. If the current description is restricted to one concept and must be generalized to cover a new example involving a second concept then perhaps a third concept can be found that is a generalization of the first two.

   e.g. the concept 'shape(X,rectangle)'

   with the new positive example 'shape(part6,triangle)'

   generalizes to the new concept 'shape(X,polygon)'

   if 'polygon' is a generalization of 'rectangle' and 'square'.

This list of generalization rules is by no means complete. Often valid generalization heuristics are known in a particular domain that are not expressed readily using any of the preceding rules. In a simple example from analytical integration that appeared in [14], it was found that when integral powers of trigonometric functions are involved, it is useful to have the concept of odd and even integers. In this particular domain, closing intervals is still a useful generalization rule but when exponents of trigonometric functions are involved, the elements in intervals must be restricted to being either odd or even integers.

2.7. Specialization Rules

Not only must a learning algorithm be able to generalize tentative concept descriptions to include positive instances, often it is also necessary to specialize a concept, restricting its scope to exclude negative examples. Specialization rules are heuristically motivated operations that are applied to reduce the scope of a concept description. The following specialization rules are mentioned by Dietterich and Michalski [6].

1. **Introduce an Exception:** The existing concept description is specialized by adding a supplementary condition that explicitly excludes known negative examples from the scope of the concept.

   e.g. the concept 'has_wings(X) **and** flies(X)'

   with the new negative example 'has_wings(ostrich) **and** (**not** flies(ostrich))'

   specializes to the new concept '(has_wings(X) **and** flies(X))
   except name(ostrich)'

   Vere has developed a learning algorithm that uses an extension of this specialization rule with one or more nested levels of generalized exception terms called 'counterfactuals' [24,25].

2. **Use a Generalization Rule in Reverse:** Specialization rules can be generated by applying generalization rules in the reverse direction. For example, a concept expressed in logic could be specialized by adding a condition (c.f. Drop Conditions rule).

   e.g. the concept 'tall(X) **and** dark(X) **and** handsome(X)'

   with the new negative example 'tall(george) **and** dark(george) **and**
   handsome(george) **and** poor(george)'

   specializes to the new concept 'tall(X) **and** dark(X) **and**
   handsome(X) **and** rich(X)'

2.8. Domain Specific Knowledge

Researchers in machine learning have discovered that general (i.e. domain-independent) learning algorithms tend to fall victim to combinatorial explosions. It is difficult to distinguish the plausible concepts from amongst the large number of generalizations and specializations that can be generated from the same example data. A deeper understanding of the properties of a domain is required if learning is to be computationally efficient. There are two main ways in which domain-specific knowledge can be introduced into a learning system:

1. **Change the Description Language:** One strategy is to change the representation language so that the domain-independent generalization and specialization rules operate efficiently. Domain knowledge can be used to prearrange constants into generalization hierarchies. Appropriate variable types can be provided. Only the relevant operators need be retained in the general description language.
2. **Add Domain-Specific Heuristics:** Another strategy is to provide heuristics that apply within an existing general-purpose description language. These heuristics might take the form of recommending that a particular structure of concept description be given special consideration. Similarly, other concept description structures may be identified as having low likelihood.

It seems that humans use other more sophisticated techniques that are not readily encoded in the description language or as separate heuristics. For example, the ability to recognize analogies between concepts from different domains is often a powerful way of making 'inductive leaps' to the most likely concept candidates. Humans are able to develop 'insight' about domains that gives them the ability to assess the plausibility of proposed concepts. Unfortunately, the prospects of modeling 'insight' and 'analogy' in computer programs remains a distant hope.

2.9. Constructive Induction

It is often the case in AI problems that the secret to constructing an efficient solution is to use an appropriate representation. For human programmers, the selection of the representation may require some experimentation or may require insight derived from similar problems encountered previously. The problem of learning by example is no exception.

Since a learning program cannot be guaranteed *a priori* to have the optimal representation for all potential learning problems, it would be advantageous for it to have the ability to change its representation language to suit the problem. For example, it might be able to add new descriptors and delete others. New logical connectives that prove to be useful might be incorporated into a representation language. Existing operators may be found to be of little use and might then be removed.

Constructive induction can pose difficult problems for a learning algorithm. Many current algorithms are data-driven systematic searches of fixed concept description spaces. Changes in the description language would be difficult to accommodate since they would change the space as it was being searched. Model-driven or mixed strategies would probably be disrupted less severely. They make no attempt to keep track of the unqualified eligibility of each and every concept candidate in the concept description space. They are already equipped with mechanisms for identifying and rejecting the current tentative candidates in favour of better candidates. If new descriptors prove to be useful, they will simply start appearing in tentative concept descriptions with no catastrophic impact on the completeness of the algorithm.

Another potential problem with constructive induction is that it may become crucial to the general acceptance of learning programs that the concepts they use be easily understood by humans. Many of the anticipated uses of intelligent system require close interaction with humans. Computer-generated description spaces may be unacceptable if the descriptors are unfamiliar to the system's users.

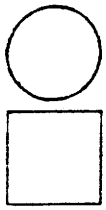
3. Representative Systems in Detail

3.1. Winston's Ph.D. thesis (1970)

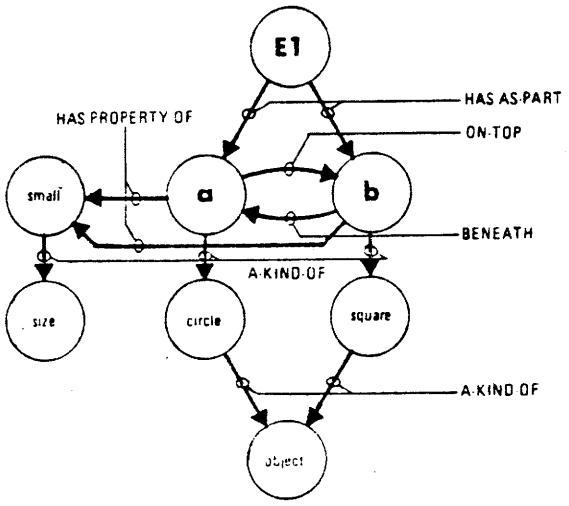
In his doctoral thesis, Winston describes one of the most influential of the early learning algorithms [27,28]. The problem studied is that of finding a maximally specific conjunctive concept description from a series of simple geometric figures. Winston devised an algorithm that performs a data-driven depth-first heuristic search of the space of concept descriptions. The coverage of the current best concept description is extended when necessary to match all positive examples by the application of generalization rules. Only one of the possibly many generalizations is retained as the best concept description: any alternatives are stored on a backtrack list. Negative examples cause the current best concept description to be specialized when they are mistakenly matched. If a concept cannot be successfully generalized, the algorithm backtracks and pursues one of the alternate concept descriptions that were generated previously but that were not pursued as the best candidate. All negative examples must be stored to allow newly activated concepts to be checked for over-generality.

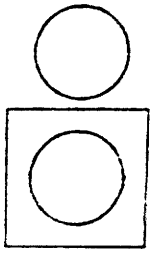
Winston's algorithm uses a semantic network as the knowledge representation language. It is used to represent the input examples supplied by the teacher, the concept descriptions developed internally by the learning algorithm and domain-specific knowledge. Examples and concept descriptions are comprised of one or more parts. In Winston's representation, each example or concept has a root node with HAS-AS-PART arcs going to nodes representing parts. Parts in turn can have unary attribute values or can be involved in binary relations with other parts. An attribute is represented by an arc from the part node to the appropriate system-wide attribute node. Attribute nodes are linked together into a system-wide domain-dependent generalization hierarchy by A-KIND-OF arcs. A binary relation is represented by two oppositely directed arcs joining the two part nodes (e.g. ON-TOP and BENEATH arcs). Each binary relation type also has a unique system-wide node that can be linked with other associated relation nodes. In Winston's examples, the ON-TOP and BENEATH nodes are joined by NEGATIVE-SATELLITE links. Other link types are required for use in the graph matching and generalization steps of the algorithm.

It is useful to consider an example that appeared in [6]. Two simple geometric figures E1 and E2 are shown below along with their semantic net representations.

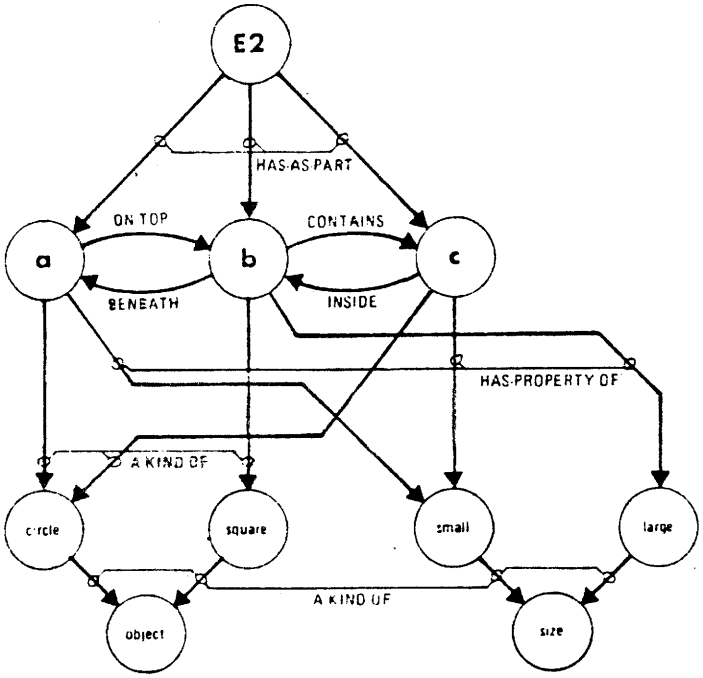


E1

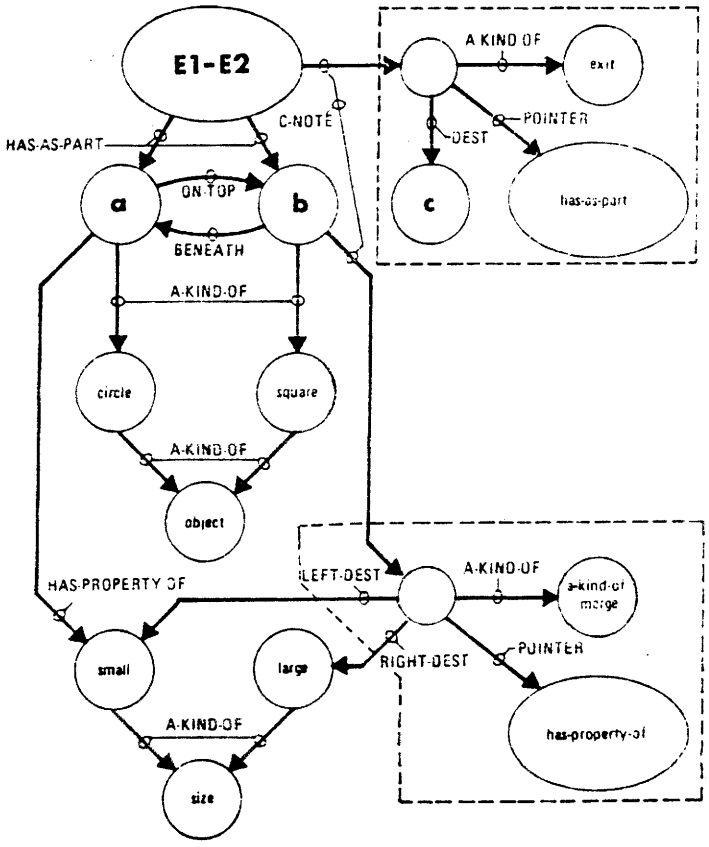




E2

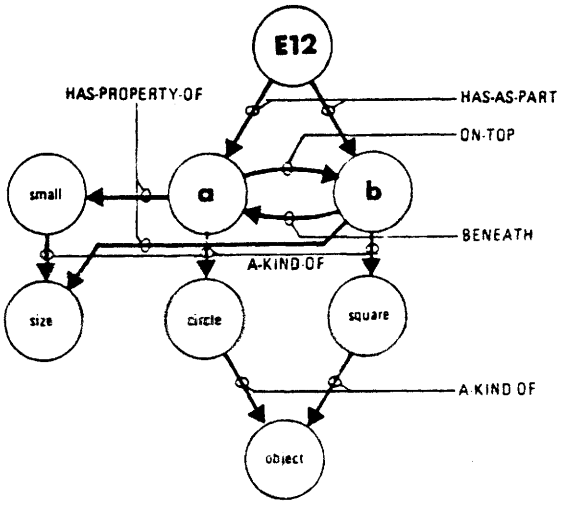


Winston's algorithm initializes the first concept description to be the description of the first positive example. Further positive and negative examples are considered serially by a two step process. In the first step, a 'best' match is found between the current concept description and the new example. An annotated difference description network showing matches and mismatches is generated. In the second step, generalizations or specializations are produced from the difference description depending on whether the example is positive or negative respectively. The annotated difference description produced by considering E1 and E2 as two positive examples in a simple learning problem is shown below:



The two boxed-out portions of the net are comparison notes. They record the mismatches between the structural descriptions of E1 and E2, namely, the different sizes of part 'b' and the unmatched part 'c' present only in E2.

By using Winston's generalization operators to resolve the mismatches, the following concept description for E1 and E2 (labeled E12) results:



Roughly translated into English, the concept reads 'there is a small circle on top of a square.'

The generalization rules 'Drop Conditions', 'Turn Constants into Variables', 'Climb a Generalization Hierarchy' and the specialization rule 'Introduce an Exception' are all implemented in Winston's algorithm as graph operations on the nets. Of the one or more generated candidates, one 'best' new concept description is chosen and all alternatives are stored on a backtrack list. Negative examples are also stored so that concept descriptions recovered from the backtrack list at a later point can be specialized to reestablish consistency.

Winston concluded that, based on his experience with his learning algorithm, learning systems would have to rely heavily on the cooperation of an intelligent teacher. He also emphasized the usefulness of negative examples that are similar to the desired concept, so-called 'near-miss' examples. In retrospect, these findings were largely a result of the characteristics of his data-driven depth-first algorithm rather than due to generally applicable principles of learning. His training sets had to be chosen carefully so that the heuristics in his algorithm, such as the choice of the 'best' concept, would efficiently find a consistent concept description. Constructing a good sequence of examples is a difficult problem requiring detailed knowledge of the learning algorithm. Winston's conclusion that the representation language must have an appropriate set of attributes has been recognized as being an important issue for all learning programs.

Advantages:

1. Simple data-driven control strategy.
2. Uses an elegant semantic net representation scheme.
3. Widely published and well known.

Disadvantages:

1. Needs a cooperative teacher, otherwise performance plummets.
2. The algorithm's choice of 'best' concept description is vulnerable to noise in the examples.
3. Order of presentation affects the final learned concept.
4. Needs a fixed hand-crafted representation language.
5. Disjunctive concepts are not representable.
6. Negative examples must be stored.
7. Alternative concept descriptions must be stored.
8. Backtracking is required.
9. Computationally expensive graph comparisons required.
10. No guarantee that the most specific generalization will be learned. The algorithm is not provably correct.

3.2. Vere : Multilevel Counterfactuals (1975-80)

In the middle 1970's, Vere studied the formal correctness of several breadth-first learning algorithms. The multi-level counterfactual, a generalization of Winston's **must-not** exception operator, was an important new idea to emerge from this work. Vere's learning algorithms find one maximally specific concept description that matches all members of a set of positive instances and no members of a set of negative instances. In several papers, Vere proposed extensions to his original algorithm to include the learning of production rules and disjunctive concepts [24,25]. A variant of the disjunctive algorithm presented in [25] restricted to learning conjunctive concepts is considered below.

Concepts in Vere's original representation scheme are conjunctions of LISP-style list structures representing logical literals. In later work, he allowed concepts to be disjunctions of literal lists. The first term in a literal is by convention the relation name.

Subsequent terms are either constants or universally quantified variables. For example, the concept of a stack of three cubes could be described as:

(cube X)(cube Y)(cube Z)(on X Y)(on Y Z)

where 'X', 'Y' and 'Z' are variables and 'cube' and 'on' are relation names. Conjunction operators are assumed between each literal. A positive example of this concept might be:

(cube a)(cube b)(cube c)(on a b)(on b c)

where 'a', 'b' and 'c' are constants denoting three specific cubes in the world.

Each conjunct can have an associated counterfactual, an exception condition that must be true for all negative examples and false for all positive examples. In form, a counterfactual consists of a conjunct and optionally, another counterfactual. An example of a concept that has a single-level counterfactual is:

(on X Y)(cube Y)(red Y) **except** (green X)(sphere X)

All positive instances must be matched by (on X Y)(cube Y)(red Y) but not matched by (green X)(sphere X). All negative examples must be matched by (green X)(sphere X). The English translation of this would be 'an object X is on top of a red cube and X is not a green sphere'. An example of a concept with a two-level counterfactual is:

(on W Z)(cube W)(plate Z) **except**
((small W)(yellow W) **except** (dense W))

In English this concept might be described as 'a cube W is on top of a plate and it is not true that W is small and yellow and W is not dense.' It is often difficult for humans to grasp the meaning of counterfactuals beyond the first level as this simple example may have demonstrated to the reader.

Conjuncts and their counterfactuals can be nested to arbitrary depth, thus the expression multi-level counterfactual. A first level counterfactual can be thought of as a first order correction to an overly general conjunct (i.e. one that covers negative instances). A second level counterfactual corrects for over-correction by the first level counterfactual: the first correction could have over-specialized the whole concept by uncovering positive examples. Levels of correction for over-specialization and over-generalization can be added to arbitrary depth to ensure consistency with all examples. The positive and negative examples themselves are represented with simple conjuncts of

literals with no counterfactuals.

Vere's multi-level counterfactual algorithm initially creates a first conjunctive concept description that covers the given fixed set of positive examples. Then, using the sets of positive and negative examples and the first concept, a second concept induction problem is constructed whose solution is the first level counterfactual. Essentially, the negative examples covered by the first concept have the matched literals in their description removed creating the positive examples for a second learning problem. The second problem's negative examples are produced by subtracting the matched literals in the first problem's positive examples. Further problems are generated recursively until the current concept is consistent: i.e. it matches all of the current positive examples but none of the current negative examples. Vere has proven that this recursive algorithm must terminate if a consistent concept description exists [25].

The individual steps in Vere's algorithm are illustrated below in a learning problem from a simple 'block's world' [25]. A consistent concept description for the four positive and four negative instances is sought.

P1.1 : (on t1 t2)(sphere t1)(green t1)(cube t2)(green t2)
P1.2 : (on t3 t4)(pyramid t3)(blue t3)(cube t4)(green t4)
P1.3 : (on t5 t6)(cube t5)(yellow t5)(cube t6)(green t6)
P1.4 : (on t7 t8)(on t8 t9)(cube t7)(green t7)(cube t8)(red t8)(cube t9)(green t9)

N1.1 : (on t10 t11)(sphere t10)(blue t10)(cube t11)(green t11)
N1.2 : (on t12 t13)(sphere t12)(green t12)(cube t13)(blue t13)
N1.3 : (on t14 t15)(on t15 t16)(cube t14)(yellow t14)(cube t15)(blue t15) &
(cube t16)(green t16)
N1.4 : (on t17 t18)(cube t17)(blue t17)(cube t18)(green t18)

The first learning problem produces the overly general concept:

(on X Y)(cube Y)(green Y)

Note that it matches negative examples N1.1, N1.3 and N1.4. That it is overly general is not too surprising since only positive examples P1.1 to P1.4 were considered. The second level learning problem has the following positive and negative examples:

P2.1 : (sphere X)(blue X)
P2.2 : (on t14 X)(cube t14)(yellow t14)(cube X)(blue X)
P2.3 : (cube X)(blue X)

N2.1 : (sphere X)(green X)
N2.2 : (pyramid X)(blue X)
N2.3 : (cube X)(yellow X)
N2.4 : (on t7 X)(cube t7)(green t7)(cube X)(red X)

The concept (blue X) matches all three positive examples but also matches negative example N2.2 so another residual learning problem must be solved. The third level problem has one positive example and three negative examples:

P3.1 : (pyramid X)

N3.1 : (sphere X)
N3.2 : (on t14 X)(cube t14)(yellow t14)(cube X)
N3.3 : (cube X)

The concept (pyramid X) does not match any of the third level negative examples so that the overall learning process halts. The final two-level counterfactual for the original problem is thus:

(on X Y)(cube Y)(green Y) **except** ((blue X) **except** (pyramid X))

Vere's inner algorithm for generating a concept from a set of positive examples has four steps. It is itself a data-driven breadth-first search for a maximally specific conjunctive concept description. The concept is initialized to the first example and then generalized to match the remaining examples by applying the 'Drop Conditions' and 'Turn constants into Variables' generalization rules. First, a set is generated of all matching literals between the current best concept and the new example. Two literals match if they have at least one common constant in corresponding positions and the same total number of terms. For the second step, subsets of the matching literal pairs found in the first step are created subject to the condition that a literal from the concept and the example can only appear once in each subset. In the third step, the subsets of matching pairs are augmented with pairs of indirectly matching literals selected from the literals not already in the subset. Two literals match indirectly if they have the same number of terms and at least one corresponding pair of mismatched constants

also appears in one position in a directly matching literal pair. Finally, in the fourth step, new candidate generalizations are produced by merging literal pairs. Differing constant terms in the same position are matched by creating new variables.

In his algorithms, Vere's goal was to develop provably correct induction algorithms to place work on learning concepts from examples on a more theoretically secure basis. While this has been achieved, there remain doubts about their practical usefulness in real AI systems. It is not entirely clear how Vere's programmed implementations avoided the combinatorial explosions that plague breadth-first search algorithms. It is suspected that heuristics must have been used [6]. A drawback of counterfactual descriptions raised by Vere is that humans can only comfortably comprehend single-level counterfactual concept descriptions [25]. It is not clear how domain-specific information could be exploited. The possibility of setting up and using generalization hierarchies among attribute values and relations was not mentioned.

Advantages:

1. Works with concept learning and production rule learning.
2. Can be extended to disjunctive concepts.
3. Has a provably correct learning algorithm.

Disadvantages:

1. Requires a fixed representation language.
2. Domain-specific knowledge is not used.
3. Breadth-first search can be combinatorially explosive.
4. Concept descriptions are often difficult for humans to understand.

3.3. Buchanan : Meta-DENDRAL (1977-78)

The learning system Meta-DENDRAL was developed by Buchanan and others at Stanford University as a knowledge acquisition tool for an existing knowledge-based system [1,2]. DENDRAL is a widely used expert system for interpreting molecular structure from mass spectrogram data. Its knowledge base consists primarily of fragmentation rules that specify how molecules break apart in a mass spectrograph, an apparatus used to identify chemical compounds. Inferring these rules from experimental data is a laborious process even for a human expert. Meta-DENDRAL was designed as a special-purpose learning program that could be used interactively by a chemist to infer new

fragmentation rules given compounds and their measured mass spectrograms.

The nature of Meta-DENDRAL's domain, noisy measurement data and a huge space of possible rules, necessitated a model-directed learning strategy. A weakly directed exhaustive search would produce excessively long computation times. Instead, generally applicable heuristics about the fragmentation process were used to guide an iterative generate-and-test search toward a stronger set of rules sufficient to explain the known empirical data. A semantic net scheme was chosen for representing compounds, nodes and arcs denoting atoms and bonds respectively. Fragmentation heuristics imposed restrictions on which kinds of bonds were most likely to break and on what kinds of atomic rearrangements could occur in the resulting fragments. For example, it could be assumed that molecules can only break once into two fragments, and that the break can only occur at a single bond. The desired goal concept description was a compact set of specific fragmentation rules operating on the known atomic structure of the compound. In this way, learning effort was directed toward the most promising, physically plausible rules.

The learning process proceeds as follows. A set of fragmentation rules operating on the given molecular structure is generated to meet the heuristic constraints and cover most of the major peaks in the mass spectrograph (i.e. the positive instances). Then the set of rules is compacted, modified and merged in an effort to improve coverage of the positive evidence and reduce any negative coverage (e.g. prediction of spectrograph peaks where none appear). This process is repeated until the fit between the measured data and the data predicted by the rule set agree to within predetermined criteria.

Meta-DENDRAL learns a disjunctive set of production rules. The rules of generalization used were 'Drop Conditions', 'Turn Constants into Variables' and 'Add Internal Disjunction'. Specialization was achieved by applying the generalization rules in reverse.

Meta-DENDRAL was particularly successful in its specialized domain. Known fragmentation rules were rediscovered from real mass spectrogram data [2]. In addition, previously unknown rules were discovered and later accepted by human specialists in the field. The program's ability to infer valuable rules in a noisy real-world domain with a proficiency equal to that of human experts has made Meta-DENDRAL an excellent example of the model-driven paradigm.

Advantages:

1. Effectively exploits domain-specific knowledge.
2. Handles noisy data.
3. Learns disjunctive and intersecting rules.

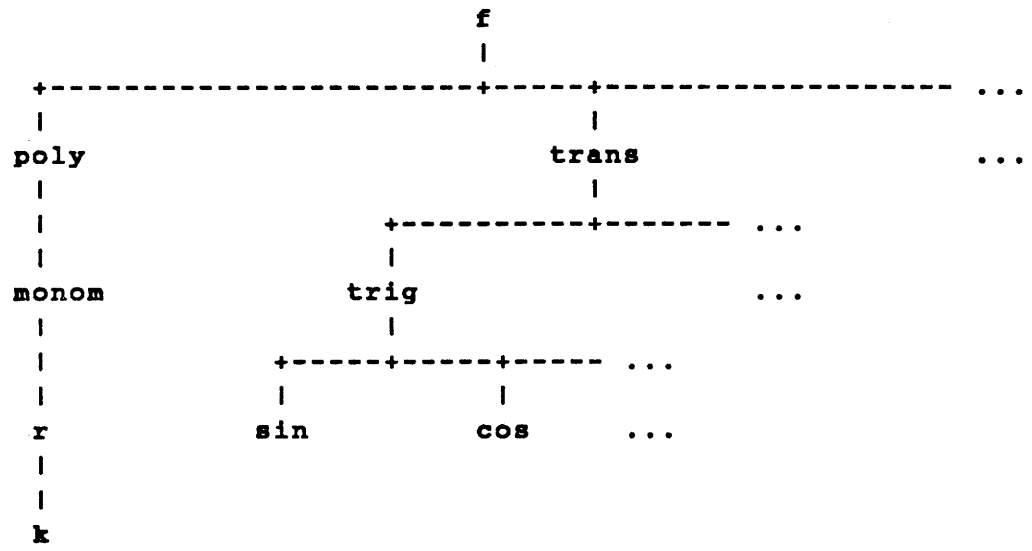
Disadvantages:

1. Essentially a domain-specific algorithm.

3.4. Mitchell : Candidate Elimination (1978)

In his doctoral thesis, Mitchell presents theoretical analysis and two practical applications of his candidate elimination learning algorithm [12]. The most important new idea that emerges from this work is that of the version space, an efficient way of representing the entire space of concept descriptions that are consistent with the training set. Candidate elimination has a simple data-driven control strategy: each time a new example becomes available, the current version space is updated by removing all concept descriptions that are not consistent with the example. A unique concept description has been found if the version space contracts to include only one element.

The version space technique can only be used for certain types of representation languages. The language's fixed sets of constants and relations are assumed to be partially ordered in generality by domain-dependent constraints. In his lecture given at IJCAI-83 Mitchell gave the following as a fragment of the generalization hierarchy that was used in his LEX learning program [14]:



where: f = real-valued function

poly = polynomial

trans = transcendental function

monom = monomial

trig = trigonometric function

r = real constant

k = integer constant

Concept descriptions must take the form of conjunctions of relations over constants and variables. The learning algorithm needs to be provided with a predicate that tests whether one given concept description is more general than another. Also required is a predicate that tests whether an instance is matched by a given concept.

By exploiting partial ordering in the space of concept descriptions, it is possible to use a compact representation for potentially large version spaces. Mitchell shows that only the most specific and most general boundaries are required to fully specify a version space [12]. These boundaries can be represented efficiently as sets of concept descriptions. Typically, the sizes of the so-called S and G sets (for the most specific and most general boundaries respectively) are much smaller than any enumeration of all members in a version space. One constraint must always hold among the members of S and G: each element in S must be less general than at least one element in G, and each element in G must be more general than at least one element in S. This restriction ensures that the version space will be properly delimited.

Mitchell's algorithm uses a positive training instance to initialize the version space. The S set is initialized to contain the instance's description and the G set is initialized to its most general possible form. It is known at this point that all potentially consistent concept descriptions must lie within this initial version space. Further examples can only reduce the size of the space by causing the S and G boundaries to move monotonically toward one another. The version space is updated indirectly by operations on S and G. When a positive instance, Ipos, is matched by all elements of S, no changes are required because all concepts in the version space must also match Ipos. If at least one element in S does not match Ipos, the S boundary needs to be generalized to remove overly specific concept candidates from the version space. Breadth-first generalization of S is achieved by applying one or more of the rules 'Drop Conditions', 'Turn Constants to Variables' and 'Climb a Generalization Hierarchy' within the constraint imposed by the G set. Finally, any elements of G that are not generalizations of elements in the new S must be deleted. The procedure for handling negative examples is analogous with the roles of S and G reversed and with the G set possibly being specialized.

The candidate elimination algorithm yields several important benefits. Until a unique concept description has been found, the version space can be used as the representation for a partially learned concept. As such, it can be used to help recognize instances. If an instance is not matched by all concepts in the version space (i.e. all elements in S fail to match) then it cannot be an instance of the concept being learned. If an instance is matched by all members of the version space (i.e. all elements in S match) then it can safely be classified as belonging to the concept. Finally, if some but not all members of the version space match an instance, then an estimate can be ventured as to the likelihood of its belonging to the partially learned concept. The version space can also be used to decide which new examples would produce the most rapid convergence of the version space. An optimal next example is chosen to be consistent with only half the concepts in the version space. A series of instances meeting this criterion causes the version space to shrink in size exponentially. If the S and G sets empty, the version space no longer has any candidates. From a correctness result proved by Mitchell, it is certain that no consistent concept description for the training set exists [12].

The candidate elimination algorithm with its use of version spaces was an important advance in machine learning. Its systematic search of all possible concept descriptions facilitates the use of incompletely learned concepts. Mitchell has used the version space method in his LEX series of learning systems [14]. Extensions to the basic algorithm for disjunctive concepts and for training sets with limited inconsistency are discussed in Mitchell's dissertation. It is not clear how the restrictive conditions imposed on the structure of the learning space will limit the usefulness of the technique.

Advantages:

1. All consistent concept descriptions are found.
2. Partially learned concepts are usable.
3. Learning is independent of the order of presentation.
4. Past examples need not be stored.
5. Backtracking is not required.

Disadvantages:

1. Need a fixed and sufficient description language.
2. No exception operator.
3. Inconsistent training examples are expensive to handle.
4. Cannot exploit domain-specific concept likelihoods.
5. Partially learned concepts are difficult for humans to understand.

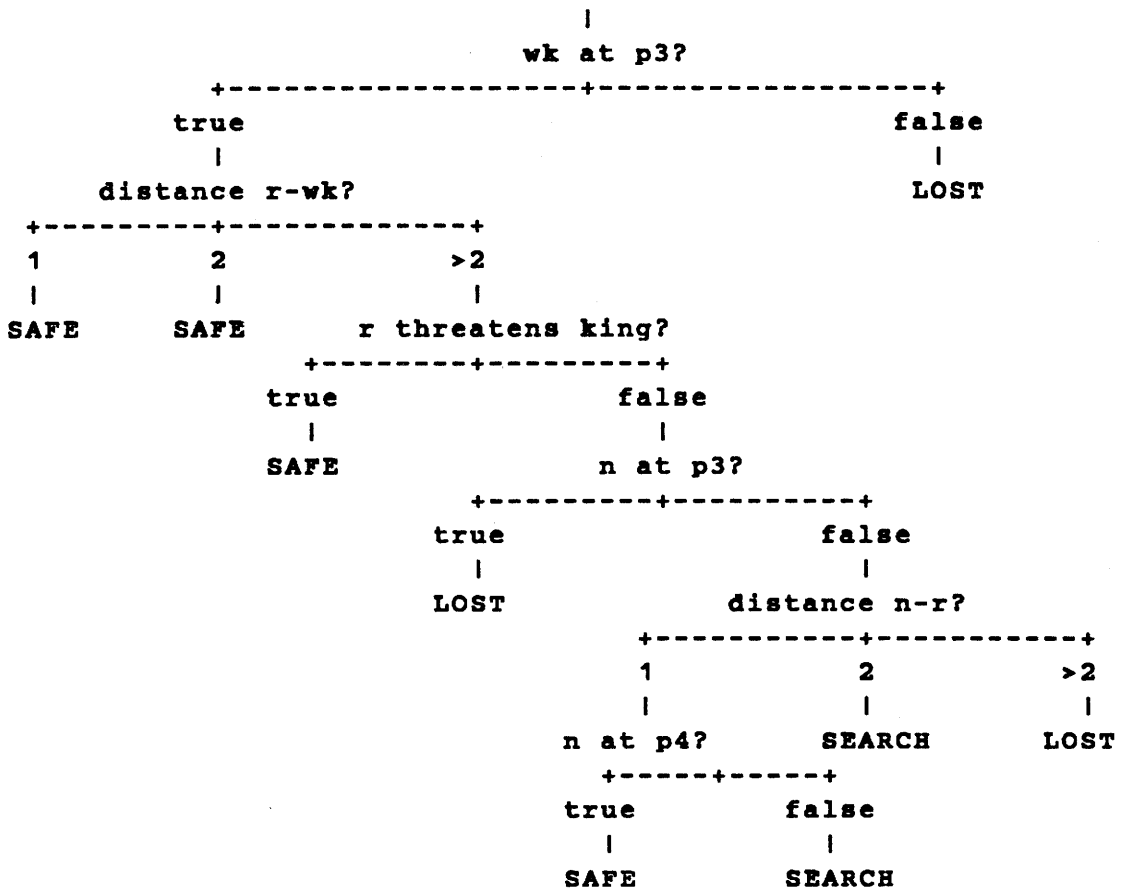
3.5. Quinlan : Chess Endgame Analysis (1979)

In the late 1970's, Quinlan designed a series of programs that learned discriminant concepts for evaluating board positions for a well known chess endgame problem [15,16,17]. Specifically, the game scenario under study was white king and rook playing against black king and knight with black about to make a move. The learning goal was to decide the game's outcome, either safe or lost for black, over the next two moves. This learned knowledge was to be expressed as a decision tree using the values of relevant features in the board positions. Quinlan chose chess as the domain because the game has already been studied extensively by human experts: the choice was not a reflection of any serious limitations in his algorithm. The motivation for the work was the observation that human experts seem to be able to identify relevant features in complicated situations and then combine them in apparently simple ways to yield

expert judgements. However, when asked to justify their reasoning, expert chess players are often unable to express the complete set of rules that they are using. Quinlan's program objectively infers a sufficient set of rules using the experts' own attributes.

Rather than attempting to classify board positions in terms of four pairs of coordinates, considerable data compression is realized by classifying board descriptions instead. In general many distinct positions have the same description. For example, there are over 11 million possible 'black to move' positions for the particular endgame that was studied (i.e. 64 x 63 x 62 x 45). Using any set of attributes reduces the number of instances to roughly 1.8 million distinct descriptions simply by removing symmetric variants from consideration. For one set of 25 attributes used by Quinlan, the database was further reduced to 29,236 instances [16].

Conventional chess wisdom was used to select attributes such as 'distance from white king to rook = 1 , 2 or more squares' or 'black king can move adjacent to the knight = true or false'. Some difficulty was encountered in finding sets of attributes that mapped only positions with identical outcomes to the same description. A set of attributes that does not have this property cannot possibly be used to form a complete decision tree. Quinlan was able to circumvent the problem by adding a new node type 'search' for those descriptions where the attributes are not sufficient for deciding the outcome. If the number of 'search' nodes is small the decision tree should be sufficient for most board positions. A fourth node type 'null' was used as a label for descriptions that did not correspond to any legal board positions. By settling for a mostly complete decision tree, only 11 nodes in a 393 node tree produced in one run were labeled 'search' rather than 'lost' or 'safe' [16]. A small portion of this tree is given below (subtree #7 in [16]):



where: 'wk at p3' means white's king is on an edge two squares from a corner
'distance r-wk' is the distance from white's rook to white's king
'r threatens k' means white's rook threatens black's king
'n at p3' means black's knight is on an edge two squares from a corner
'distance n-r' is the distance from black's knight to white's rook
'n at p4' means black's knight is on a diagonal one square from a corner

Quinlan's learning algorithm operates as an iterative generate and test cycle. A random subset of the complete set of board descriptions is chosen as the first 'window'. Using heuristics that promote compactness, a decision tree rule is generated that partitions all elements in the window so that all descriptions reaching the same node from the root have the same outcome. Then the generated rule is tested for accuracy against the entire set of pre-classified board descriptions. If the rule correctly classifies all descriptions, the algorithm terminates. Otherwise, a new window is formed from the old window and incorrectly classified board descriptions, and the cycle is repeated until a correct rule is found.

The behaviour of Quinlan's program is encouraging in several respects. Convergence on the correct decision tree tends to be rapid. Using only a small fraction of the total number of descriptions in the window, a rule is found that applies to them all. Summarizing the behaviour of the algorithm over several endgame problems, Quinlan proposed the rule:

running time = constant * (total size of the instance space)
* (number of description attributes)
* (complexity of decision tree)

In other words, Quinlan's algorithm learns rules in time linearly proportional to the number of examples. This is in marked contrast to the combinatorial explosions to which Winston's and Vere's algorithms can fall victim. On the negative side, some difficulty was occasionally experienced in finding sets of attributes sufficient to resolve all board descriptions into classes with common outcomes. This suggests that either human experts are not fully aware of all the attributes that they are using, or that they are in fact relying on a combination of pattern analysis, search and perhaps other methods.

As a footnote to this problem, after considerable trail and error Quinlan was able to develop a set of 23 binary-valued attributes that partioned the set of descriptions into sets with common outcomes [17]. The total space of 1.8 million board positions collapsed to just 428 distinct board descriptions. The exact decision tree that resulted contained only 83 nodes. Interestingly, the attributes in this run were not nearly as obvious as the set of 25 in the initial study. They consisted of broader, less obvious patterns of piece positions that are not recognized as primitive features in the chess literature.

Advantages:

1. Efficiently learns decision trees for large databases using small windows.
2. Human knowledge of relevant features is exploited to infer the unexpressed rules underlying expert performance.

Disadvantages:

1. Needs a sufficient set of domain-dependent attributes.
2. A complete set of examples needs to be pre-classified by independent means.

3.6. Michalski and Stepp : Conceptual Clustering (1980-83)

At the University of Illinois (Urbana-Champaign), Michalski and Stepp developed algorithms that construct hierarchical classifications based on what they called conceptual clustering [9,10]. Traditionally, the problem of identifying clusters in data is assisted by using numerical techniques such as those of pattern analysis and numerical taxonomy. Clusters in data are recognized as sets of elements that are close together in some predefined multidimensional feature space. Efficient computer programs are available that implement various different clustering criteria to form numerically optimal classifications. Michalski and Stepp observed that humans have difficulties interpreting such clusters: they are unlike the kinds of clusters that humans normally use. They pointed out that object classes normally are chosen according to a few important features. As a result, the meaning of each cluster tends to be readily apparent from the description itself. Conceptual clustering assumes that clusters should be expressed as simple conjuncts of descriptive relations. This should make their meaning readily apparent. Creating a good classification involves finding a set of conjunctive concept descriptions that optimizes the partitioning with respect to such criteria as fit, simplicity and disjointedness. The programs CLUSTER/paf and CLUSTER/2 were written to embody these ideas.

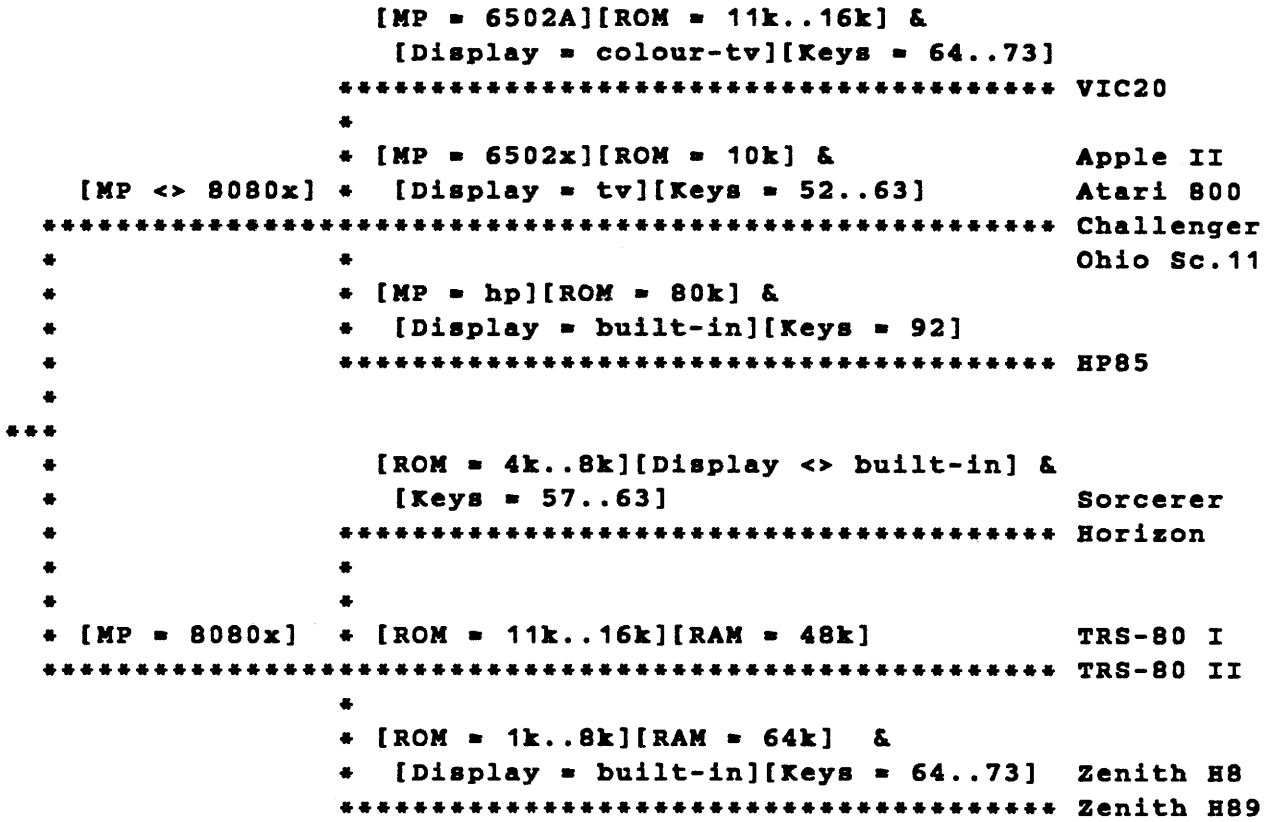
The variable valued logic VL1, developed by Michalski, was used as the representation language for cluster concept descriptions [8]. Variables in VL1 are typed to finite, discrete value sets such as unordered sets, linearly ordered sets, and tree-structured sets. Concepts are conjunctions of relational statements. A relational statement is comprised of a variable and a subset of its possible values. Thus the following expression is a concept comprised of four relational statements:

[Height = tall][Colour = {blue,red}][Length > 2][Weight = 2..5]

A logical AND is assumed between each relational statement. The four variables in the expression are 'Height', 'Weight', 'Length' and 'Weight'. The second relational statement illustrates the internal disjunction operator: the variable 'Colour' can assume either one of the values 'blue' or 'red'.

The classifications created by the CLUSTER programs are hierarchical taxonomies. A hierarchical taxonomy is recursively defined as either a single terminal node enumeration of instances, or as a concept description and a disjunction of one or more sub-hierarchies. For example, CLUSTER/2 was used to generate the following

classification for a universe of twelve microcomputers [10]:



The terminal node comprised of the Sorcerer and Horizon microcomputers belongs to the major subdivision [MP = 8080x] and the minor subdivision [ROM = 4k..8k][Display <> built-in][Keys = 57..63]. It should be pointed out that the choice of divisions is governed by the choice of some clustering quality evaluation function. Changes in this function would likely cause substantial changes in the form of the classification.

The learning algorithm used in CLUSTER/2 consists of a clustering module and a hierarchy building module. The former finds a partitioning for the set of examples that is optimal with respect to a list of criteria chosen by the user. Among the criteria that might be chosen are, the closeness of the fit between the examples and the concept classes, the syntactic simplicity of class descriptions, the number of classes, the disjointness of the classes with respect to one another and the syntactic similarity of the examples grouped together into the same class. The optimal clustering can be found by systematically generating all possible partitionings and choosing the one that best meets the chosen criteria. As is, this approach is not acceptable for practical problems because of the combinatorially explosive number of possible partitionings. Heuristic

techniques were applied successfully in CLUSTER/2 to keep the run times reasonable. The details of how a disjoint set of partitions is created are involved and are not repeated here. Briefly, seed events are picked to start each class. Then various generalization and specialization rules are employed to 'grow' the classes to include all of the remaining examples. The hierarchy building module applies the clustering module at each level for each class to construct the overall classification. It terminates once the fit between the members of the proposed new classes and their class descriptions is no longer an improvement over the fit achieved at the previous level.

The classifications produced by CLUSTER/2 were compared against the clusters produced by the numerical taxonomy package NUMTAX in two domains: the classification of popular personal computers and the reconstruction of a classification of selected plant diseases [10]. It was found that the classifications produced by conceptual clustering corresponded much better to the human-developed classifications for the same domains. This result should not be too surprising since CLUSTER/2 used the same sorts of attributes as those used by humans, qualities that are not considered by purely numerical techniques. The more understandable classifications produced by conceptual clustering do incur an increased computational cost. For example, in the microcomputer problem 4 to 40 seconds of processor time was required on a Cyber 175 compared with the 60 milliseconds required for each classification produced by NUMTAX. This difference is caused by the iterative nature of CLUSTER/2's generate and test algorithm. The increased one shot cost may not be significant in many applications since conceptual clustering should save time at the interpretation stage.

Advantages:

1. Produces human-understandable classifications.
2. Cluster classes have easily understood meanings.

Disadvantages:

1. Need a domain-dependent evaluation function for cluster quality.
2. Classes must be describable by conjunctive descriptions.
3. Computationally expensive compared with numerical taxonomy.
4. Relative importance of features not specifiable.
5. Substructure of objects not exploited as a clustering feature.

4. Critical Discussion

The following sections identify areas of weakness in current algorithms that learn concepts from examples.

4.1. Monotonic Learning

A seemingly unavoidable characteristic of human learning is that the accumulation of new knowledge rarely proceeds smoothly. Some misconceptions must be expected along the way. Still, in the longer term, humans eventually recognize and correct faulty aspects of their accumulating understanding. The inability to handle short term setbacks is an area of weakness in current learning algorithms, particularly those that use the data-driven control strategy. Most of these programs embody the optimistic assumption that the concept description will improve monotonically. For example, it is common to encode all of the known positive examples into the evolving concept description(s). It is assumed that there will be no need to re-examine past data to deal with inconsistencies that might emerge. Without separately storing past examples, it is impossible to keep track of certainty, assumptions, dependencies and other related details. Such information is helpful when recovering from misconceptions.

4.2. Intolerance of Errors and Noise

Most learning algorithms, especially those that are data-driven, are unable to tolerate errors, noise or inconsistent training sets. For example, Winston's and Vere's algorithms would be disrupted completely by a single spurious example. Mitchell's algorithm can only be modified to handle moderate inconsistencies with substantially increased computational cost [12]. Noise is inevitable in the real world. If learning algorithms are to be able to handle real world input, they will have to be able to tolerate background levels of noise. Intolerance of errors and noise is closely related to the often overly optimistic assumption of monotonicity. A learning algorithm must be able to recover from misconceptions caused by error in previous data since it cannot always be assumed that error will be detected when an example is first considered. Being able to store past examples independently of the concept descriptions would probably simplify error recovery since it would permit knowledge to be reconstructed from original data. It would also allow the algorithm to estimate the level of noise in its source of further examples.

4.3. Exploiting Domain Knowledge

Learning programs such as Buchanan's Meta-DENDRAL and Quinlan's chess endgame analyzer owe much of their performance to their ability to exploit domain-specific knowledge. General learning algorithms are important to study but it is their ability to access and use domain knowledge that is a major factor in their power [4]. Using domain knowledge, an algorithm can direct most of its resources toward the most likely concept description models. Using domain knowledge, errors in the training set can be recognized earlier. Also, by using domain knowledge the more plausible generalizations and specializations can receive more attention than the often large number of possible syntactic mutations. Model-driven learning algorithms are in a better position to exploit knowledge than those of the data-driven control strategy. To make domain-specific knowledge more usable, new knowledge representation schemes should be developed.

4.4. Unjustified Generalization and Specialization

Current learning algorithms are not able to justify the ways in which concept descriptions can be most usefully generalized and specialized. This is a drawback in several respects. Typically, numerous syntactic mutations can be proposed by the available generalization and specialization rules. Without knowing the justifications for each rule, a learning algorithm must expend equal effort on each possibility, making combinatorial explosion a problem. Also, it is possible that the justification for using a particular rule might be enhanced or diminished by knowledge acquired during past experience. This information would also be useful in speeding up the rate of learning. Finally, if the results of learning algorithms are to be accepted by humans, explanations and justifications for them will be expected.

4.5. Constructive Induction

Most current learning algorithms use a fixed representation language. It is assumed that concepts will be both expressible and learnable in a previously chosen concept description space. This assumption can be problematic: a good set of descriptors and operators is not always known ahead of time. Humans often have to discover the best representations before efficient learning can take place. It has been the experience in AI that the performance of many programs is critically dependent on the choice of representation language. For these reasons, constructive induction is an important

capability that has not appeared in many learning algorithms.

4.6. Disjunctive and Intersecting Concepts

To date, most learning algorithms have only been able to learn conjunctive concept descriptions. A few, such as CLUSTER/2 and Meta-DENDRAL, were designed to handle disjunctive concepts. Meta-DENDRAL is also capable of learning intersecting concepts: a peak in a mass spectrogram can be attributed to more than one fragmentation rule. In general though, learning disjunctive and intersecting concepts is more difficult than learning purely conjunctive concepts. Learning such concepts tends to be a non-monotonic process requiring domain-specific heuristics and iteration. Interestingly, humans seem to be able to learn disjunctive and intersecting concepts apparently without substantially increased conscious effort. Perhaps humans are able to use deeper knowledge to prune away all but the most likely of the possible partitionings of the examples.

4.7. Fuzzy and Probabilistic Concepts

Most of the knowledge representation schemes used by current learning programs can only handle exact, deterministic information. Yet, there is much information in the world that is not readily encodable within these restrictions [19]. Much knowledge is inherently vague or uncertain. Fuzzy and probabilistic knowledge representation schemes have been developed for AI programs that must manipulate non-exact knowledge but these have not yet appeared in machine learning algorithms.

4.8. Empirical Methodology

AI research has a tradition of considering various different domains using rather ad hoc techniques and often not attempting to rigorously test assumptions [20]. The justification for this empirical approach is that the field is still in a primitive state and that investigators should be free to try a variety of methods in diverse domains to build up an understanding of the important issues. Also, AI problems tend to have so many parameters that it would be impractical to try to systematically control them all. However, there is a danger inherent in current practice. Algorithms and data structures are rarely published at the source code level, just described in English prose. Such informal descriptions can be ambiguous, making it hard for other investigators to duplicate and extend published results. Winston's algorithm requires an intelligent teacher to achieve

reasonable learning performance, but how then will this algorithm be compared with others in the same domain? How will a teacher's helpfulness be controlled or compared? As was seen in the introduction to this essay, there is no universal agreement as to what constitutes learning behaviour. Progress in machine learning would probably accelerate if agreement could be reached on standardized learning problems that could then receive more systematic investigation.

5. Directions for Future Research

The following sections discuss directions for further research with programs that learn concepts from examples.

5.1. Model-driven and Mixed Control Strategies

More research on the model-driven and mixed control strategies is justified on the basis of past experience. Many of the problems identified in the critical discussion such as error tolerance, non-monotonic learning, use of domain knowledge, constructive induction, disjunctive and intersecting concept learning, would be more readily tackled using top-down methods rather than any of the well known data-driven algorithms. Techniques must be found that address the problem of having to store all past examples so that they can be used later to check the consistency of proposed concepts. Quinlan's chess program demonstrated that only a small representative subset of all possible examples need be stored when a rule valid for the complete set is to be learned. Perhaps only the most 'typical' or 'representative' positive instances and exceptions for the current rules should be remembered. Heuristics that indicate the best examples to retain are required.

5.2. Coping With Modifiable Concept Description Languages

The use of a single representation language for both the examples and the concept descriptions can be a drawback. There might be advantages to using a low-level representation for stored examples as opposed to a more highly encoded representation used for the concept descriptions. Thus changes in the concept description language would not cause problems if past examples needed to be re-interpreted using a new set of descriptors. Learning programs that have separate representation schemes and that allow the concept description language to be changed are worth investigating.

5.3. Standardized Domains and Learning Problems

In the interests of making the testing and comparison of different learning algorithms more systematic, standardized domains and learning problems should be found. Learning does not seem to be a behaviour that can be tested fairly in any one problem. The winner of the decathlon series of events in track and field is often considered to be the best all-round athlete. By analogy, a series of sample learning problems could be selected to determine the best all-round learning algorithm. The mere exercise of

creating such a series of test problems and then getting them accepted by the research community would help clarify what is understood to be learning behaviour. Even an imperfect test series would provide a definite goal toward which new algorithms could be developed.

5.4. Learning Appropriate Generalization and Specialization Rules

Not knowing which generalization and specialization rules are most plausible in a given learning situation is a weakness in current algorithms. Learning programs should be developed that learn over the course of many learning problems which rules are plausible and which are unlikely. Perhaps generalization and specialization rules should be constructed as condition-action production rules in which the condition part embodies the learned limits of a rule's usefulness.

5.5. Bottleneck Detection and Active Learning

Mitchell points out that a learning algorithm should have some declarative awareness of the performance of its different components [14]. If a bottleneck is discovered, an algorithm should be able to request specific kinds of training examples from the environment and then make improvements to its own internal mechanisms. This would reduce the dependence that some of the current algorithms have on a benevolent external teacher. There are limits as to how well a human can learn in a read-only mode. Two-way interaction with the training source (e.g. a teacher) allows a student to request clarification on matters that do not make sense. Similarly, learning programs should be devised that actively participate in the learning process.

5.6. A Final Word of Caution

Finally, it may be that significant progress in machine learning will have to wait for progress in other areas such as knowledge representation, learning by discovery, probabilistic and fuzzy reasoning, associative storage structures and parallel hardware. Even if this pessimistic scenario proves correct, additional successful learning programs like Meta-DENDRAL will likely emerge as spin-offs of machine learning research. This possibility by itself is justification enough for further efforts in this difficult area of AI.

6. References

[1] Buchanan, B.G. and Mitchell, T.M., 'Model-Directed Learning of Production Rules', Computer Science Department, Report #STAN-CS-77-597, (Stanford University, Stanford, CA, 1977).

[2] Buchanan, B.G. and Feigenbaum, E.A., 'DENDRAL and Meta-DENDRAL : Their Applications Dimension', Heuristic Programming Project Memo 78-1, Computer Science Department, Report #STAN-CS-78-649, (Stanford University, Stanford, CA, 1978).

[3] Buchanan, B.G., Mitchell, T.M, Smith, R.G. and Johnson, C.R., 'Models of Learning Systems', v.11, pp.24-50 in J.Belzer, A.G. Holzman and A. Kent (Eds.), *Encyclopedia of Computer Science and Technology* , (Marcel Dekker Inc., New York, 1978).

[4] Carbonell, J.G, Michalski, R.S and Mitchell, T.M., 'Machine Learning Part I: A Historical and Methodological Analysis', Department of Computer Science, Carnegie-Mellon University, Report #CMU-CS-83-135, (Carnegie-Mellon University, Pittsburgh, 1983).

[5] Carbonell, J.G, Michalski, R.S. and Mitchell, T.M., 'An Overview of Machine Learning', pp.3-24 in R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), *Machine Learning : An Artificial Intelligence Approach* , (Tioga Press, Palo Alto, CA, 1983).

[6] Dietterich, T.G. and Michalski, R.S., 'Inductive Learning of Structural Descriptions: Evaluative Criteria and Comparative Review of Selected Methods', *Artificial Intelligence* , v.16, 1981, pp.257-294.

[7] Lenat, D.B. and Brown, J.S., 'Why AM and EURISKO Appear to WORK', *Artificial Intelligence* , v.23, 1984, pp.269-294.

[8] Michalski, R.S., 'Pattern Recognition as Rule-Guided Inductive Inference', *IEEE Transactions on Pattern Analysis and Machine Intelligence* , v.PAMI-2, no.4, July 1980, pp.349-361.

[9] Michalski, R.S. and Stepp, R., 'Revealing conceptual structure in data by inductive inference', pp.173-196 in J. Hayes, D. Michie and Y-H Pao (Eds.), *Machine Intelligence 10* , (Ellis Horwood, Chichester, England, 1981).

[10] Michalski, R.S. and Stepp, R.E., 'Automated Construction of Classifications: Conceptual Clustering Versus Numerical Taxonomy', *IEEE Transactions on Pattern Analysis and Machine Intelligence* , v.5, no.4, July 1983, pp.396-410.

[11] Michalski, R.S. and Stepp, R.E., 'Learning From Observation: Conceptual Clustering', pp.331-364 in R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds.), *Machine Learning : An Artificial Intelligence Approach* , (Tioga Press, Palo Alto, CA, 1983).

[12] Mitchell, T.M., 'Version Spaces: An Approach to Concept Learning', Ph.D. dissertation, Computer Science Department, Report #STAN-CS-78-711, (Stanford University, Stanford, CA, 1978).

[13] Mitchell, T.M., 'Generalization as Search', *Artificial Intelligence* , v.18, 1982, pp.203-226.

[14] Mitchell, T.M., 'Learning and Problem Solving', Computers and Thought Lecture, *Proceedings of the 7th International Joint Conference on Artificial Intelligence* , 1983, pp.1139-1151.

[15] Quinlan, J.R., 'Induction over Large Data Bases', Stanford Heuristic Programming Project, Memo HPP-79-14, Computer Science Department, Report #STAN-CS-79-739, (Stanford University, Stanford, CA, 1979).

[16] Quinlan, J.R., 'Discovering Rules by Induction from Large Collections of Examples', pp.168-201 in D. Michie, (Ed.), *Expert Systems in the Micro Electronic Age* , (Edinburgh University Press, Edinburgh, 1979).

[17] Quinlan, J.R., 'Semi-autonomous acquisition of pattern-based knowledge', pp.159-172 in J. Hayes, D. Michie and Y-H Pao (Eds.), *Machine Intelligence 10* , (Ellis Horwood, Chichester, 1981).

[18] Quinlan, J.R., 'Learning Efficient Classification Procedures and their Application to Chess End Games', pp.463-482 in R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds.), *Machine Learning : An Artificial Intelligence Approach* , (Tioga Press, Palo Alto, CA, 1983).

[19] Rich, E., *Artificial Intelligence* , (McGraw-Hill, New York, 1983).

[20] Ritchie, G.D. and Hanna, F.K., 'AM: A Case Study in AI Methodology', *Artificial Intelligence* , v.23, 1984, pp.249-268.

[21] Scott, P.D., 'Learning: the Construction of A Posteriori Knowledge Structures', *Proceedings of 3rd National Conference on Artificial Intelligence* , (AAAI, 1983), pp.359-363.

[22] Scott, P.D. and Vogt, R.C., 'Knowledge Oriented Learning', *Proceedings of the 7th International Joint Conference on Artificial Intelligence* , 1983, pp.432-435.

[23] Simon, H.A., 'Why Should Machines Learn?', pp.25-38 in R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds.), *Machine Learning : An Artificial Intelligence Approach* , (Tioga Press, Palo Alto, CA, 1983).

[24] Vere, S.A., 'Induction of Concepts in the Predicate Calculus', *Proceedings of the 4th International Joint Conference on Artificial Intelligence* , 1975, pp.281-287.

[25] Vere, S.A., 'Multilevel Counterfactuals for Generalizations of Relational Concepts and Productions', *Artificial Intelligence* , v.14, 1980, pp.139-164.

[26] *Webster's New Collegiate Dictionary* , (G. & C. Merriam Co., Springfield MA, 1981).

[27] Winston, P.H., 'Learning Structural Descriptions from Examples', pp.157-209 in P.H. Winston (Ed.), *The Psychology of Computer Vision* , (McGraw-Hill, New York, 1975).

[28] Winston, P.H., 'Learning Class Descriptions from Samples', ch.11 in *Artificial Intelligence* , 2nd Ed., (Addison-Wesley, Reading, MA, 1984), pp.391-414.

7. Appendix A - Glossary

Candidate Elimination - a data-driven learning algorithm discussed by Mitchell in his doctoral thesis [12]. A version space is updated with each positive and negative example to remove inconsistent concept description candidates.

Combinatorial Explosion - the situation that exists when the amount of computation grows rapidly with the size of the problem (e.g. exponentially increasing).

Concept Description - a symbolic data structure that represents a concept (i.e. for a particular domain describes one class of instances, or a classification of all known instances).

Counterfactual - 'a set of conditions which must be false if a generalization is to be satisfied' [25].

Deductive Inference - a mode of reasoning in which postulated facts are recognized as being logical consequences of a known body of facts and rules.

Generalization - the process of extending the scope of the current description to include more instances. Also, the new concept description whose scope has been extended.

Inductive Inference - a mode of reasoning in which a collection of facts is analyzed and general theories or rules that imply the facts are postulated.

Martin's Law - 'you cannot learn anything unless you almost know already' [28]

Maximally Specific Conjunctive Concept Description - A concept description that is the logical AND of one or more predicates and that matches the least number of possible instances beyond the known positive examples.

Near-miss - a negative example that is similar to previously encountered positive examples of the concept.

Negative Example - a counter-example of a concept that may result in specialization of the current concept description.

Partially Ordered Set - A set on which is defined a transitive binary relation. Let 'A', 'B' and 'C' be elements and '>' denote the relation, If (A > B) and (B > C) are true then (A > C) must also be true. The binary relation is not necessarily defined for all pairs of elements.

Positive Example - a correct instance of a concept that may result in generalization of the current concept description.

Predicate - a labeled relationship that possibly holds among one or more terms in a logical system.

Specialization - the process of narrowing the scope of a description to reduce the set of instances that it matches.

Term - a logical object that is recursively defined as either a variable, a constant, or a functional expression made up of a functor name and one or more terms.

Version Space - a subset of the space of all possible conjunctive concept descriptions that is delimited by most specific and most general boundary descriptions.