On Integrating Inductive Learning with Prior Knowledge and Reasoning

A Dissertation
Presented to the
Department of Computer Science
Brigham Young University

in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

by
Christophe G. Giraud-Carrier

December 1994

Acknowledgments

I thank the many people who made this work possible. Most faculty members in the Department of Computer Science had some impact, either directly or indirectly on my perspective of research. Dr. Tony Martinez provided insightful feedback and the resources for the completion of the work, and gave me many opportunities to present it in various conferences. Dr. David Embley was always willing to lend an ear to my ideas, both in research and regarding other matters of graduate study. Dr. Bill Hays gave me insightful feedback on the content of the research. Drs. John Higgins and Douglas Campbell kept my appetite for mathematics and theory alive. Dr. William Barrett was always a good person to bounce ideas of off and get regular "reality checks" from. Dr. Scott Woodfield was an example of enthusiasm about research ideas and of an almost insatiable curiosity. Drs. David Wright and James Cannon of the Department of Mathematics were my topology mentors, and though not directly involved in my research, always showed interest in my progress and expressed support and encouragement.

I am particularly indebted to my wife and children for their constant support and encouragement. They are the joy of my life. There were many a days of frustration when my dear wife picked me up and rekindled in me the desire to go on.

Most of all, I am thankful to a Heavenly Father who sees fit to give me the opportunities that are mine, and without whom none of this would have been possible.

This research was funded by the Department of Computer Science at Brigham Young University, and by grants from Novell Inc., and WordPerfect Corp.

This dissertation, by Christophe Giraud-Carrier is accepted in its present form by the Department of Computer Science of Brigham Young University as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Tony Martinez, Committee Chairman

David Embley, Committee Member

Bill Hays, Committee Member

Date

David Embley, Graduate Coordinator

On Integrating Inductive Learning with Prior Knowledge and Reasoning

Christophe G. Giraud-Carrier
Department of Computer Science
Ph.D. Degree, December 1994

ABSTRACT

Learning and reasoning are both aspects of what is considered to be intelligence. Their studies within AI have been separated historically, learning being the topic of neural networks and machine learning, and reasoning falling under classical (or symbolic) AI. However, learning and reasoning share many interdependencies, and the integration of the two may lead to more powerful models. This dissertation examines some of these interdependencies and describes several models, culminating with a system called FLARE (Framework for Learning And REasoning). The proposed models integrate inductive learning with prior knowledge and reasoning. Learning is incremental, prior knowledge is given by a teacher or deductively obtained by instantiating commonsense knowledge, and reasoning is non-monotonic. Simulation results on several datasets and classical commonsense protocols demonstrate promise.

COMMITTEE APPROVAL:

____
Tony Martinez, Committee Chairman

____
David Embley, Committee Member

____
Bill Hays, Committee Member

____
David Embley, Graduate Coordinator

Table of Contents

CHAPTER 1
Introduction

Machine Learning has recently emerged as a promising alternative to programming for the implementation of solutions to many problems on computers. Whether the approach is inspired by biological plausibility (e.g., artificial neural networks) or by psychological plausibility (e.g., classical artificial intelligence), the goal is to design systems capable of learning and reasoning about certain tasks such as analogy and classification. The learning approach offers many advantages over the programming approach, particularly in contexts where information is uncertain or incomplete. Indeed, programming presupposes that all of the knowledge necessary for implementing the solution of a given problem is available *a priori*. This assumption is valid in many well-defined, closed domains, such as mathematics or accounting. However, there are many situations in which the very nature of the knowledge or the representation thereof is incomplete and uncertain, such as medical diagnosis, pattern recognition, natural resource management, and many decision-making applications such as loan underwritting and mine-rock discrimination. Whereas solutions to problems in these domains are difficult to program, learning may offer a viable and effective alternative.

Section 1 of this chapter reviews the basic principles of inductive learning, and discusses the need for, source of, and use of prior knowledge in such systems. Section 2 addresses the issue of (commonsense) reasoning. Section 3 looks at some of the interdependencies between learning and reasoning. The existence of such interdependencies and the resulting attempt at integrating learning and reasoning into a unified framework form the basis of this dissertation. Section 4 summarizes sections 1 through 3 into a single thesis statement, and section 5 gives the outline of the dissertation.

1. Inductive Learning and Prior Knowledge

Induction is the process of going from the particular to the general. It is often considered to be a mechanism for knowledge acquisition or learning. Inductive learning (IL) has been the subject of much research leading to the design of a variety of algorithms. Excellent surveys of many of these algorithms are in [MIC83b, SHA90].

In general, IL systems generate classification rules from examples. Typically, the system is first presented a set of hopefully representative examples (objects, situations, etc.), also known as a *training set*. Examples are usually expressed in the attribute-value language, and represent recorded instances of attribute-value pairs together with their corresponding classification. Attributes represent properties relevant to the context of the application to be learned. For example, when learning to classify shapes as either rectangles, squares, or triangles, some useful attributes may be length, width, and number of sides.

A conjunction of attribute-value pairs, where each attribute appears at most once, is called a *feature*[1]. Given a training set, an IL system's goal is to discover sets of sufficient critical features or rules, that correctly classify the examples of the training set (convergence), and adequately extend to previously unseen examples (generalization).

Though machines are still a far cry from matching human qualitative inductive leaps, IL systems have proven useful over a wide range of applications in medicine (e.g., breast cancer and hepatitis detection), banking (e.g., credit screening), defense (e.g., mine-rock discrimination), botany (e.g., iris variety detection, mushroom identification), and other areas (see, for example [HAZ94, KUV94, SEJ87]). They are practical and useful when data from past experience is available and can be engineered into a suitable training set. In general, human interaction is limited to the gathering and compilation of examples. As research on autonomous agents continues, this process itself may eventually become automated. Given an application A, inductive learners typically extract the critical features of A from the training set, without human intervention. Empirical studies show that good results can be achieved with such inductive systems (see, for example [KIB87, MOO89, QUI86, SEJ87, ZAR94]).

However useful, IL has several important drawbacks. Many such systems (e.g., backpropagation) are typically slow as they may require many passes over the training set to achieve a reasonable level of performance. Also, there is no guarantee that given an arbitrary training set, the system will find enough good critical features to get a reasonable approximation of A. Indeed, since critical features are extracted from training examples, the system's ability to generalize is dependent upon the representativeness of its training set. Moreover, the number of sets or conjunctions of candidate critical features that must be searched is exponential in the number of inputs, and IL becomes computationally expensive [BLU87]. Finally, IL does not take advantage of an important source of knowledge, namely prior knowledge.

Human learning is often not the sole result of exposure to random examples. Rather, built-in mechanisms (e.g., pain), and teachers (present in most social structures, such as the family or the school) account for much of humans' ability to efficiently learn complex problems. In this work, we refer to teacher-given rules as *precepts*. A precept reflects some high-level-knowledge (or common sense) about the real-world. It "suggests something advisory and not obligatory communicated typically through teaching" (Webster's Dictionary). Precepts are identical to critical features except that they are given by a teacher (including a machine) and not derived by the inductive learning algorithm. Also, they are not necessarily correct, but convey some hint or bias. For instance, when dealing with how weather conditions affect what one wears, a possible precept is: "if it rains, take an umbrella," regardless of the temperature, the day of the week, or any other unnecessary (don't-care) detail. The following two examples illustrate the interplay between examples and precepts.

Example 1 - Hot Stove

[1] Such conjunctive features represent only a subset of the possible features. Other features involving for example relationships between attributes may also be defined. For example, in distinguishing between squares and rectangles, a useful feature is the equality or inequality of length and width. The more complex the features, the more difficult it is to design a learning algorithm capable of generating them.

> Regardless of most external circumstances, it does not make sense to put your hand on a red-hot stove. However, rather than acquiring this general rule through successive examples of settings involving your hand on the stove, as a human, you are exposed to precepts taught by others (e.g., "it will burn you"), which, together with a built-in pain mechanism, allow the (almost instantaneous) learning of the rule.

Example 2 - English as a Second Language

> Foreign-language students first learn English grammar rules (e.g., the plural of a noun is formed by tagging an *s* to the end of the singular form of that noun), and are then exposed to exceptions (e.g., geese instead of gooses, mice instead of mouses).

In Example 1, it is the given precept that governs learning. Examples are essentially irrelevant in this situation. Example 2 however shows that precepts may not always be sufficient. In that situation, once the precept is learned from a teacher, it is exposure to examples and exceptions that completes the learning. Similarly, there may be learning scenarios where no information other than raw examples is available. Hence, it is a major premise of this dissertation that learning should involve both examples and precepts that constitute prior knowledge.

Much effort has recently been expended in understanding the sources and use of prior knowledge in learning [DES93]. The strong knowledge principle [WAT86], early work on bias [MIT80], and the scarcity of interesting positive theoretical results [VAN94] suggest the difficulty of learning without sufficient *a priori* knowledge. That is, examples should indeed be augmented by prior (domain) knowledge. Such knowledge can be obtained from an expert or teacher, or it can be automatically generated as a result of instantiating commonsense knowledge. It may be given to the system interactively or *a priori*, and can take on many different forms (see, for example [BUN90]).

One way to use prior knowledge is as an initial learning bias, whose purpose is both to prune the input space and to constrain the search for correct generalizations. If such prior knowledge is expressed in terms of rules, then the system is able to deal with both *intension* and *extension*. The rules given *a priori* are the intension and may embed knowledge about the attributes and their relationships to the target classification. The examples in the training set are the extension, from which rules based on experience (rather than prior knowledge) may be generated. Experience can then be combined with theory to yield a system capable of confirming, refuting, and modifying its "innate" knowledge based on empirical observations.

One of the contributions of this dissertation is the design of models that effectively combine inductive learning with one special form of prior knowledge, called *precepts*. Previous work [GIR93a, b] demonstrates the increase in predictive accuracy due to the use of precepts given by a teacher. The models presented here go further by discussing other uses of precepts (e.g., combination of intensional and extensional approaches) and providing mechanisms to automatically generate them from available commonsense knowledge expressed in first-order logic clauses.

One further contribution in the area of purely inductive learning is the design and evaluation of an efficient metric for heterogeneous learning spaces. In particular, the metric allows the proposed algorithms to handle applications with continuously-valued attributes. It also accounts for rules containing don't-cares, and is further extended to the handling of missing attributes. Empirical results suggest that the new metric is not only adequate, but also widens the range of applications that can be effectively handled.

Finally, the models proposed here are incremental. Typically, IL systems assume that all of the knowledge relevant to the learning task is available (*a priori*) in the training set. We claim that this assumption may not always be reasonable, and is certainly not (psychologically) plausible. Human learning is naturally incremental. Humans learn over time. A complete or even only representative set of examples is seldom available *a priori*. Rather, examples are seen over time, with some occuring more often than others, and some being spurious or noisy. Adaptation to newly acquired knowledge is an essential part of human learning. This self-adaptivity resulting from incrementality is a critical issue in the design of learning autonomous agents.

2. Commonsense Reasoning

The form of reasoning we wish to refer to here consists of a combination of deduction (i.e., rule-based reasoning) and analogy (i.e., similarity-based reasoning). Deductive processes typically do not result in the generation of new facts, rather they establish cause-effect relationships between existing facts. Deduction may be applied *forward* by seeking the consequences of certain existing hypotheses, or *backward* to discover the necessary conditions for the achievement of certain goals.

The study of deductive reasoning goes at least as far back as the early Greek philosophers, such as Socrates and Aristotle. Its formalization has given rise to a variety of logics, from propositional to first-order predicate logic to default logic to several non-monotonic extensions. Many of these logics have been successfully implemented in artificial systems, such as the logic programming language PROLOG and a wide variety of expert systems (e.g., MYCIN for medical diagnosis, PROSPECTOR for geological prospection). Description of several systems and logics are in [BRE91, COR84, GIN87, LUK90, WAT86, WEB81].

Analogical, or similarity-based reasoning can beneficially be used to complement deductive reasoning. In particular, when no rule exists that exactly matches a new situation, analogy may be used to attain a result. Analogical reasoning typically uses similarity with known situations to make inferences about new situations. For example, in the context of problem solving, one may use plans about previously solved problems to derive, via a series of transformations, a plan for a new, but similar, problem [CAR83]. The identification of similarities between situations varies from simple feature overlap [SUN92] to more complex theories involving both features and the relationships between them [GEN89]. An application of analogical learning and reasoning to Szechwan cooking is in [HAM89].

One particularly interesting and useful form of reasoning, that combines deduction and analogy, is commonsense reasoning [MIN94, SUN92, WOL94]. Common sense may be defined as the ability that any "functional" human being has to cope with the world, and to quickly make useful inferences in the presence of incomplete and uncertain information. The lack of such "simple" mechanisms in AI models has become known as the *brittleness bottleneck* in the AI community [LEN90, MCC84, SUN92]. Much work has been expended in characterizing common sense [FOR89, LER92, LIF88, SUN92, WOL94], formalizing its representation [DAV90, HOB85, LIF90, ZLA90], constructing logics (or implementations thereof) powerful enough to handle it [BRE91, GOL93, LUK90, REI80], and developing models dealing with it, or at least some meaningful subset of it [HOR90, YOU93, SUN92, SUN93].

One of the challenges of commonsense reasoning is its non-monotonicity. Informally, a reasoning protocol is non-monotonic if the acquisition of new information causes the retraction or invalidation of some previously accepted facts. The logics that have been proposed are typically extensions of first-order predicate logic, that use added "machinery," such as circumscription [MCC80], semi-normal defaults [REI81], or hierarchical theories [KON88], to handle non-monotonic information while essentially retaining consistency in the overall knowledge base. Another approach consists of tolerating inconsistency in the knowledge base, but providing only reasoning mechanisms that ensure that no inconsistent conclusions are ever reached. It is this latter approach that the models presented here implement. Note however that knowledge of the system's behavior (i.e., giving priority to more specific rules) and of the rules at hand allows one to revert to a first-order formulation (with negation) of the knowledge base. In that sense, the models produce results similar to [KUN94], except that there is no constructive induction and the theory is obtained incrementally.

Another contribution of this dissertation is the integration of such patterns of reasoning with inductive learning and prior knowledge. The models or logics referred to above focus on representation and reasoning, but do not support learning. The models presented here are not as general in terms of representation and reasoning power, but they start to take advantage of some of the interdependencies between learning and reasoning.

3. Learning and Reasoning: Interdependencies

Despite their differences, inductive learning and deductive reasoning are strongly interrelated. The ability to reason about a domain of knowledge is often based on rules about that domain, that should or could be learned somehow. And the ability to reason can often guide the acquisition of new knowledge, or learning.

As a learning system is being trained, it can use its acquired knowledge to *generalize* on new situations. Generalization can be viewed as a limited form of reasoning since it uses existing knowledge to infer new knowledge. In the proper context, the rules generated by (or given to) an inductive learning system can be viewed as defaults and the corresponding classifications as target conclusions. The reasoning power of such systems is then characterized as default reasoning. When default reasoning is supplemented with similarity-based reasoning (when no default applies), the system can perform useful plausible inferences, that account for several important commonsense protocols (see [SUN92], Appendix A).

One of the greatest challenges of deductive systems is knowledge *elicitation*, that is the acquisition of rules from experts. Elicitation is a tedious task that presents many difficulties both practically and theoretically. If a sufficiently rich training set can be obtained, then inductive learning may provide an alternative. It is typically easier and faster to record examples than it is to gather rules. On the other hand, if some *commonsense* knowledge can readily be obtained and instantiated to the specific domain, then deductive reasoning may also be used to improve inductive learning by, for example, automatically generating precepts. Note that rules and examples need not be mutually exclusive. A system's knowledge base can indeed be constructed from both teacher-given rules and inductively generated ones.

We do not know of any work attempting to combine induction and deduction in a unified framework. Another contribution of this dissertation is the proposal of seven desirable properties for the combination of inductive learning with prior knowledge and reasoning. These properties serve as foundation to the design of the proposed models. Also, as was mentioned above, one of the models provides at least one alternative to knowledge elicitation by deductively generating precepts from commonsense knowledge expressed as first-order logic clauses.

4. Thesis Statement

Though much effort has been devoted to understanding learning and reasoning in artificial intelligence, we do not know of any models that attempt to integrate these two complementary processes. Rather, there is a vast body of research in machine learning, often focusing on inductive learning from examples, quite isolated from the work on reasoning in artificial intelligence. Though learning and reasoning are different, they share many interdependencies, and the integration of the two may lead to more powerful models. This dissertation proposes several models that attempt to combine inductive learning with prior knowledge and reasoning into a unified framework.

5. Outline of the Dissertation

The dissertation is organized as a collection of published or submitted original papers. Each paper is prefaced by a few introductory paragraphs intended to provide a logical (or historical) link to the rest of the dissertation. For the most part, the rest of the chapters are exact copies of the papers with simple formatting changes to guarantee consistency throughout. Hence, some redundancy is to be expected. Abstracts and acknowledgments have been omitted, and references have been compiled into a single bibliography. Minor changes suggested by careful committee members and reviewers have also been made. As copyright was transferred to some publishers prior to the compilation of this dissertation, permissions to reprint have been sought where necessary from the rightful owners of the copyright.

Chapters 2 and 3 are preliminary. They are historically significant however, and are helpful in situating the research in a broader, longer-range perspective. Chapters 4 through 8 expose the main results of the research. A brief overview of each chapter is given below.

Chapter 2 focuses on ASOCS' Adaptive Algorithm 1 (AA1) [MAR86, MAR88], and presents an analysis of the convergence and generalization of AA1 [GIR92]. Though not a main focus of this research, AA1 provided some of the foundational ideas of the dissertation. In particular, AA1 is an incremental learner that combines rules with examples. In chapter 2, AA1 is briefly reviewed, a detailed proof of its correctness is given, and empirical evidence of its generalization is reported. The proof, though interesting in its own right, may be skipped without hindering understanding of the dissertation. Some of AA1's limitations are also discussed. Solutions to these limitations represented our first step in the direction taken by the research presented here.

Chapter 3 reviews work based on my Master's thesis [GIR93a]. It introduces the notion of precept-driven learning, where precepts are a generalization of AA1's rules, and provide an important foundation for the integration of default reasoning with inductive learning [GIR93b]. The rationale behind the use of precepts as prior knowledge is given, and PDLA (Precept-Driven Learning Algorithm) is presented as our first attempt at overcoming some of AA1's limitations and integrating incremental, inductive learning with prior knowledge.

Chapter 4 gives a brief argument for why machines should exhibit learning abilities, and proposes seven desirable properties for artificial learning systems, namely: incrementality, non-monotonicity, inconsistency and conflicting defaults handling, abstraction, self-organization, generalization, and computational tractability [GIR94a]. The proposed set of properties is not claimed to be complete, nor does it imply that all properties must necessarily be present in all learning models. Rather, it focuses on issues that are not often explicitly addressed and provides a basic foundation for the design of more efficient algorithms. Each property is examined and we show how their use can improve learning and reasoning, and potentially widen the range of applications of artificial learning systems. The algorithms presented in this dissertation serve as proof of concept, by attempting to integrate the proposed properties into a unified framework in which inductive learning is combined with prior knowledge and reasoning to yield models capable of dealing with a greater variety of interesting problems.

Chapter 5 presents PDL2 (Precept-Driven Learning 2), the first model attempting to combine inductive learning with prior knowledge and default reasoning [GIR94b]. Upon reading some work by Sun [SUN92, SUN93] on a connectionist approach to the representation of and reasoning about knowledge, and examining PDLA in the new light shed by this work, it became apparent that PDLA could be extended with limited reasoning ability. PDLA already combines inductive learning with prior knowledge. Hence, extending it provides a useful starting place for achieving the goal of this dissertation. PDL2 is the result of these extensions. Though PDL2 does not completely integrate learning and reasoning, it provides mechanisms to combine them in a meaningful way. In particular, it accounts for several (commonsense) reasoning problems both extensionally and intensionally.

Chapter 6 is a result of needed extension to PDL2, due to the fact that many real-world applications contain a mixture of nominal and linear (or ordered) attributes. Though most learning algorithms ignore the issue and use a homogeneous metric to compute similarity between patterns, it does seem that such homogeneous metrics are inappropriate for inherently heterogeneous spaces. Chapter 6 proposes a metric that properly handles homogeneous spaces while being able to efficiently handle heterogeneous spaces [GIR94c]. The most interesting aspect of the proposed metric is its ability to handle don't-cares, as well as continuous attributes directly (i.e., no need for discretization). Impacts of the metric's definition on the combination of learning and reasoning are also discussed in this chapter.

Chapter 7 introduces an enhanced model, called ILA (Incremental Learning Algorithm) for the combination of inductive learning with prior knowledge and reasoning [GIR94d]. The new model is compared to PDL2. Its advantages include the use of a heterogeneous metric (extension of the metric proposed in Chapter 6), more potential for incrementality, greater predictive accuracy, and a somewhat simpler learning algorithm. The reasoning power remains the same as that of PDL2 however. A better characterization of ILA's reasoning is given, and several of its critical properties (see Chapter 4) are discussed in detail.

Chapter 8 is the culmination of the research. It proposes a more general framework, called FLARE (Framework for Learning And REasoning), for the integration of inductive learning with prior knowledge and reasoning [GIR94e]. FLARE supports forward chaining and the automatic generation of precepts via the instantation of available commonsense knowledge expressed in first-order logic clauses. FLARE can be instantiated both at the conceptual and subconceptual levels. The empirical studies conducted with ILA (and PDL2) are limited to "toy" problems. To further demonstrate the validity of FLARE, several more "real-world" examples are provided. A better characterization of many important concepts (e.g., generalization, conflicting defaults) is given, a mechanical way of translating first-order statements into attribute-value vectors is proposed, and many yet unsolved (or ill-addressed) issues are thoroughly discussed.

Chapter 9 concludes the dissertation. It first summarizes the major results of the dissertation, and then discusses future research. The issue of commonsense reasoning is revisited and alternative approaches, based on object-oriented modeling concepts and semantic nets, are suggested.

CHAPTER 2
Analysis of the Convergence and Generalization of AA1*

The work presented in this chapter is of historical significance to the research of this dissertation. It is indeed the study of AA1 that sparkled an early interest in the design of incremental learning systems.

An important ability, inexistent in most training set learners, but found in AA1, is the ability to use general rules (short-hand instances) in learning. In particular, one can construct training sets that reflect human learning by first presenting general rules to the system, followed by exceptions. Hence, AA1 combines inductive learning with prior knowledge in the form of pre-encoded rules. AA1 provided a useful starting point for our research, and the models presented in this dissertation, while different, retain some of its flavors.

Section 7 is not in the accepted paper. Its intent is to outline some of AA1's drawbacks that served as a basis for future research.

1. Introduction

The last decade has seen a renewed interest in connectionist computing. Some of the reasons for this interest can be traced to the discovery of the wide range of neural network applications, the increasing interest in self-organizing systems, and the design of improved learning algorithms. A good overview of the current state of the art is found in [RUM86]. Most of the current models are based on static networks of computing nodes, where learning is effected by changing the weights of the connections between nodes and/or by modifying the nodes' functions. Adaptive Self-Organizing Concurrent Systems (ASOCS) are dynamic networks that learn by adapting both the function performed by the nodes and the overall network topology. Hence, the network grows over time to fit the problem.

ASOCS operate in one of two modes: learning and execution. In execution mode, the network receives inputs and produces outputs as the data flow asynchronously and in parallel through the network. In learning mode, the system is incrementally presented a set of Boolean rules, which is called the *instance set*. As each instance is presented, the network adapts both its topology and its nodes' functions to learn the new instance, while preserving consistency with previously acquired knowledge.

Several learning algorithms have been developed for ASOCS. They include Adaptive Algorithm (AA) 1 [MAR88], AA2 [MAR91a] and AA3 [MAR91b]. AA1 is the object of this paper. AA1 learns by discrimination and implements knowledge in a distributed fashion. This paper analyzes the convergence and generalization properties of AA1.

Convergence refers to the system's ability to learn the instance set. Convergence has played a significant historical role in the development of neural networks. Rosenblatt's perceptron [ROS58], the first neural network, was quickly proven to be limited to linearly separable functions [MIN67]. However, it was also shown that a (at least) three-layer network could be used to represent any arbitrary function. The challenge was thus to design a learning algorithm for multi-layer networks. Researchers struggled until the first multi-layer learning algorithm, backpropagation [RUM86, WER74], was proposed. However, because it uses a gradient-descent technique and is thus subject to local minima, there is no guarantee, in practice, that backpropagation will find a plausible solution to a training set. The first part of this paper reviews AA1 and formally proves that AA1 converges on any arbitrary Boolean instance set. Moreover, unlike backpropagation which requires several passes over the training set, AA1 is a "one-shot" learner with fast, bounded learning time.

During learning, the system is presented with only a subset of the function it is to solve. Generalization refers to the system's ability to correctly guess the rest of the function. To be of any practical use, a system should not be confined to simply remembering (i.e., rote learning), but should also be capable of inducing new knowledge from past experience. Sections 4 and 5 of this paper discuss AA1's generalization scheme and show that it produces promising results. AA1's originality lies in the fact that it generalizes not on the basis of how closely the new unknown input matches one of the learned rules, but rather on how well the network can discriminate the unknown input from the learned features of the opposite class.

Section 2 gives an overview of AA1. Section 3 contains the proof that AA1 converges on any arbitrary Boolean instance set. Section 4 discusses other features of AA1 including generalization, and the handling of inconsistency. Section 5 presents results of simulations of AA1 on real-world data. Section 6 summarizes the paper.

2. AA1 Learning Algorithm

This section gives a short overview of AA1. Only those details essential to the proof of the following section are presented. For more details on AA1, see [MAR86, MAR88].

An *instance set* is a set of *instances*, where each instance is a conjunction of Boolean input values together with the output value they imply. We use V and V' to mean *V is positive* (i.e., true) and *V is negative* (i.e., false), respectively. The following are examples of instances:

$$AB \rightarrow Z \quad \text{(i)} \qquad\qquad B'C \rightarrow Z \quad \text{(ii)} \qquad\qquad AC' \rightarrow Z' \quad \text{(iii)}$$

The conjunctive part of instances need not contain an input value for each input variable. Such instances are used as short-hand. If I is an instance in which no value is specified for some input variable V, then I represents both instances obtained from I by adding V and V', respectively, to the conjunctive part of I. For example, if A, B and C are the only 3 input variables, then instance (i) represents both instances $ABC \rightarrow Z$, and $ABC' \rightarrow Z$. In other words, instance (i) means that when A and B are both positive, Z must become true, regardless of the values of any other input variables.

The value of the output implied by an instance is its *polarity*. Instances (i) and (ii) are positive and instance (iii) is negative. Two instances with the same polarity are said to be *concordant* with respect to each other, while two instances of opposite polarity are *discordant*. Instances (i) and (ii) are concordant; instances (ii) and (iii) are discordant.

The basic architectural unit in AA1 is a *node*. During execution, each node computes the AND or OR function of its (possibly inverted) inputs. During learning, since all inputs may not have a value in an instance, these functions are extended to accommodate unspecified (denoted ?) inputs (e.g., 0 AND ? implies 0, 0 OR ? implies ?). In addition, each node records, in the form of a *node table* (*NT*), the value it outputs for each instance in the training set. The NT has two columns, P and N, holding the output values for positive and negative instances, respectively, and a column D which is used in the node selection part of the algorithm presented below. Figure 1 shows an example of a node.

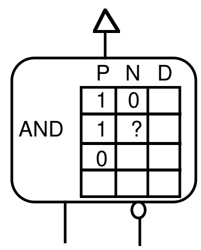


Figure 1 - Example of a Node

AA1 incrementally builds a network with a single top node whose output is the network's output. Figure 2 shows an example of such a network (for simplicity, the D columns are not represented). During execution, the node tables are ignored and the system functions as a logic network. For example, if B, C, D, and G are all positive, then the network of Figure 2 outputs 1. During learning, the node tables are used to control discrimination as explained below.

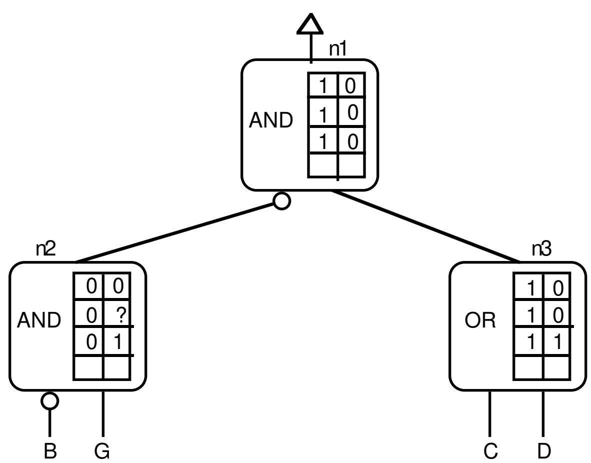


Figure 2 - Example of a Network

If a cell in the NT contains the value 0 or 1, the node *discriminates* the instance represented by that cell from all discordant instances whose cells contain the opposite value. For example, let *K* be the node n3 of Figure 2. Consider the first cell of the P column whose value is 1, and let *I* be the corresponding instance. If *K* outputs 1, there is no way to tell if it is I that is matched or if it is the instance corresponding to the third cell of the N column. However, if K outputs 0, clearly I is not matched. But K outputs 0 for the instances corresponding to the first and second cells of the N column, so K is able to discriminate between I and these two instances. Note that cells containing ?'s cannot participate in discrimination. There are four classes of nodes, based on discrimination:

- *discriminant node*: discriminates at least one positive instance from one negative instance,
- *non-discriminant node*: does not discriminate at least one positive instance from one negative one,
- *complete discriminant node*: discriminates every positive instance from every negative instance,
- *one-sided discriminant node*: asserts one value for either all positive or all negative instances, and the opposite value for at least one discordant instance (see, for example, node n2 of Figure 2).

Discrimination is the key factor in AA1. When it is presented an instance set, AA1 learns to discriminate between positive and negative instances. AA1 *fulfills* the instance set (i.e., converges) if the network it builds outputs 0 for every negative instance and 1 for every positive instance. That is, AA1 converges if and only if the final network's top node is complete discriminant and has 1's in its P column.

Learning proceeds as follows. Instances are introduced one at a time in an incremental fashion. For each new incoming instance, AA1 goes through a preprocessing phase that maintains the instance set consistent and seeks to minimize it. An instance set is inconsistent if it contains discordant instances whose conjunctive parts could be true simultaneously. For example, instances $AB \rightarrow Z$ and $BC \rightarrow Z'$ produce an inconsistency since for A, B and C positive, the first one implies that Z is positive while the second one implies that Z is negative. Inconsistency is solved by giving precedence to the newer instances (see [MAR94] for details). Complete minimization is not reasonable, but AA1 attempts partial minimization through pairwise comparison of the training instance and instances in the current instance set. The correctness of this aspect of the algorithm has been proved elsewhere [RUD91a]. The correctness of the actual learning algorithm has not. Section 3 fills this gap.

The result of the above preprocessing phase is a *delete-list* and an *add-list* containing the instances to be removed and added, respectively, from the current instance set to keep it consistent and somewhat minimal. To process the delete-list, AA1 simply causes each node to empty the corresponding cells in its NT. Each instance in the add-list is then added to the current instance set and presented to the network. Each node places its output for the new instance in the corresponding cell of its NT. The network's output is then checked. If it is concordant with that of the new instance, no changes are made to the network and the next training instance can be processed. If, on the other hand, it is discordant or ?, then the current top node is no longer complete discriminant and modifications must be made to the network. Since the current top node correctly discriminates the instance set, with the exception of the new instance, AA1 constructs a new node that discriminates the new instance from all discordant instances. This one-sided discriminant node (OSDN) is then combined with the current top node to build a new complete discriminant node. The network finally undergoes a phase of *self-deletion* in which nodes that are no longer needed (such as non-discriminant nodes) are removed from the network. Self-deletion increases parsimony. Details on the various kinds of self-deletions may be found in [MAR86, MAR88].

The construction of OSDN is effected by a process of *node selection* and *node combination*. Node selection consists of a greedy search through the network for a set of nodes that, when combined, give rise to OSDN. The nodes that are selected are those nodes that discriminate the new instance from the largest number of remaining non-discriminated discordant instances (the D column of the NT keeps track of which discordant instances are already discriminated by the selected nodes). If node selection fails, i.e., if the selected nodes would not combine to create OSDN, then *node creation* takes place, and AA1 guides the construction of new nodes that can discriminate the new instance from all remaining non-discriminated discordant instances. From each remaining non-discriminated instance an input is randomly selected that is sufficient to discriminate it from the new instance. Pairs of these chosen inputs are then used as inputs for the new nodes whose functions must be set so that they output 0 or 1 when both inputs are matched and the complement when either one is not matched. Once all the necessary *growth* nodes have been selected and/or created, they are combined so as to build OSDN. Each combination connects two nodes to a new node that discriminates the union of the discriminations done by the nodes from which it was created.

Due to space, we can give only a high-level example of how AA1 updates the network upon receipt of a new instance. A detailed example is found in [MAR88]. Consider the network of Figure 2. Assume the new instance is positive and causes the top node to output ?, node n3 to output 1 and node n2 to output ?. The result of processing the new instance is in Figure 3. Note that Node Selection took place and node n3 was selected as a growth node, since it discriminates the new instance from two of the three discordant instances. Node Addition was then necessary and resulted in the creation of input A (sufficient to discriminate the new instance from the remaining discordant non-discriminated instance). Node n3 was subsequently combined with input A (degenerated growth node) to produce a OSDN (node n4). The OSDN was finally combined with the old top node (node n1) to produce a new complete discriminant top node (node n5).

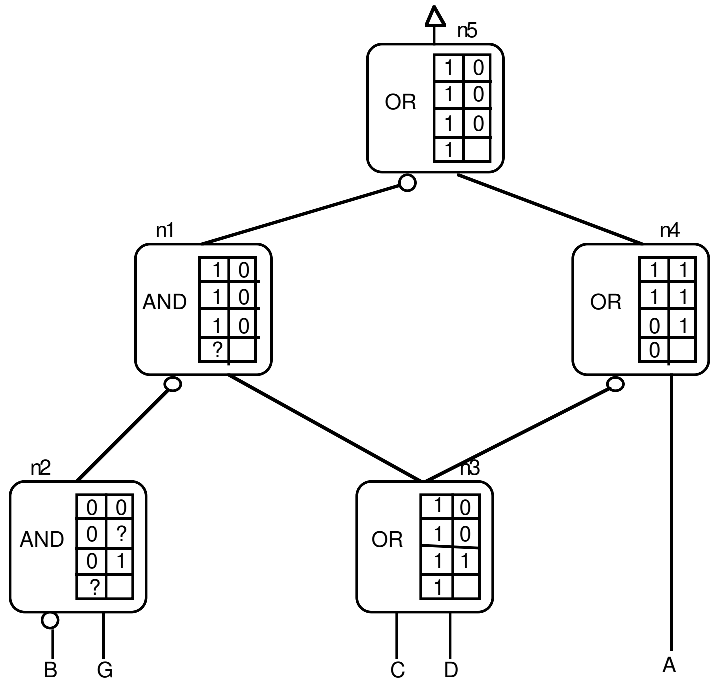


Figure 3 - Modified Network

When two nodes N_1 and N_2 combine and give rise to a parent node *PN* with function *f*, *PN*'s node table is loaded by applying *f* pairwise to corresponding cells in N_1's and N_2's node tables. It follows that *PN*'s node table contains the values that *PN* would output for each instance in the current instance set (i.e., there is no need to rebroadcast all the instances of the instance set to the modified network).

3. AA1 Is Correct

In this section, we prove that AA1 always fulfills the current instance set. Note first that processing the delete-list is trivially correct since emptying cells does not change the complete discriminant nature of the current top node. We show that processing the add-list is also correct.

Let *NI* be the instance being processed. Let *I* be the set of current discordant instances that *NI* must be discriminated from, and *OSDN* a one-sided discriminant node that discriminates *NI* from *I*. Let *CTN* be the current top node, and *NTN* the node resulting from combining *CTN* and *OSDN*.

Lemma 1 shows that Node Selection and Node Addition give rise to a set of growth nodes, each of which discriminates *NI* from some subset of *I* such that the union of these subsets is *I*. Lemma 2 shows that when any two growth nodes, discriminating *NI* from two subsets of *I*, are combined, the resulting parent node discriminates *NI* from the union of these subsets. Theorem 1, which shows that the result of Node Combination is *OSDN* follows immediately by induction. Theorem 2 then shows that AA1 fulfills the instance set after *NI* has been processed. The result follows immediately from the finiteness of the add-list.

Lemma 1:

Let $G_1,...,G_k$ be the growth nodes resulting from Node Selection and Node Addition. Let $I_1,...,I_k$ be the respective sets of instances that each G_i discriminates from *NI*. Then $I=I_1 \cup ... \cup I_k$.

Proof:

If Node Selection is sufficient, then the result follows trivially from the algorithm, since Node Selection consists precisely of recruiting nodes that each discriminate *NI* from some subset of *I* until the union of these subsets is *I*. Suppose then that Node Addition is necessary.

Let $G_1,...,G_j$ ($j<k$) be the growth nodes resulting from Node Selection alone, and $I'=I_1 \cup ... \cup I_j$. We must show that $G_{j+1},...,G_k$ resulting from Node Addition discriminate *NI* from $I''=I - I'$, or similarly that $I''=I_{j+1} \cup ... \cup I_k$. For each instance in I'', there exists at least one variable which is the complement of one of the variables in *NI* (otherwise the current instance set would be inconsistent). One such variable is chosen for each instance in I''. Pairs of these selected variables serve as inputs to $G_{j+1},...,G_k$ which are set to the AND or OR function to guarantee that their output is either 1 or 0 when both inputs are matched and the complement when either one is not matched. Hence, each added node will output one value for *NI* (one of the inputs is not matched) and the complement for the instances of I'' from which its input variables were selected (both inputs are matched). Therefore, $G_{j+1},...,G_k$ discriminate *NI* from subsets $I_{j+1},...,I_k$, respectively, of I''. Since a variable is selected from each one of the instances in I'', it follows immediately that $I''=I_{j+1} \cup ... \cup I_k$.

Lemma 2:

Let G_1 and G_2 be two growth nodes other than *CTN* and *OSDN*. Let I_1 and I_2 be the sets of current instances that are discriminated from *NI* by G_1 and G_2, respectively. Let *PN* be the parent node created when G_1 and G_2 are combined. Then *PN* discriminates *NI* from $I_1 \cup I_2$.

Proof:

G_1 and G_2 output either 0 or 1 for *NI* (? is not possible, otherwise they would not discriminate *NI* and thus could not be growth nodes). Since $PN \neq NTN$, it does not matter whether *PN* outputs 0 or 1 for *NI*. Hence, there are eight possible functions for *PN*, summarized in [2, Table 6.2]. We give the complete proof for the case where both G_1 and G_2 output 1 for *NI*. Then, the function of *PN* is the AND of the outputs of G_1 and G_2. Now, because G_1 discriminates *NI* from the instances in I_1, G_1 outputs 0 for every instance in I_1. Similarly, G_2 outputs 0 for every instance in I_2. Applying the AND function pairwise to G_1 and G_2 will thus cause *PN* to 1) output 1 for *NI* and 2) output 0 for every instance in I_1 and every instance in I_2. It follows that *PN* discriminates *NI* from $I_1 \cup I_2$.

Theorem 1:

Let $G_1,...,G_k$ be the growth nodes resulting from Node Selection and Node Addition. Let $I_1,...,I_k$ be the respective sets of instances that each G_i discriminates from NI. Let *PN* be the node obtained after the G_i's have combined. Then *PN* is *OSDN*.

Proof:

Each node that is the result of combining two growth nodes is labeled as a growth node and can therefore participate in combination with other nodes. Any pair of growth nodes may combine. Hence, the combination process happens finitely many times (exactly k-1 combinations) and halts when there is a single growth node left, namely *PN*. By induction on the result of Lemma 2, *PN* discriminates *NI* from every instance in $I_1 \cup ... \cup I_k$. But, from Lemma 1, we have $I = I_1 \cup ... \cup I_k$. It follows that *PN* discriminates *NI* from every instance in *I*, and hence, *PN* is *OSDN*.

Theorem 2:

NTN is a complete discriminant node such that AA1 fulfills the new instance set.

Proof:

There are only four ways, depending on the polarity of *NI* and the output of *OSDN* for *NI*, that *OSDN* and *CTN* may combine to produce *NTN*, as summarized in [2, Table 6.3]. We give the complete proof for the case where *OSDN* outputs a 1 for *NI* and *NI* is positive. Then *NTN*'s function is the OR of the outputs of *CTN* and *OSDN*. Since *OSDN* outputs 1 for *NI* and *NI* is positive, the N column of *OSDN* contains only 0's. Since *CTN* fulfills the old instance set, the N column of *CTN* contains only 0's. Hence applying the OR function pairwise to the N columns of *CTN* and *OSDN* will produce the N column of *NTN* that contains only 0's. Similarly, the P column of *CTN* contains 1's in every cell, except the one corresponding to *NI*. But by assumption the cell corresponding to *NI* contains a 1 in the P column of *OSDN*. Hence applying the OR function pairwise to the P columns of *CTN* and *OSDN* will produce the P column of *NTN* that contains only 1's. Consequently, *NTN* is a complete discriminant node, and since *NTN* outputs 1 for positive instances and 0 for negative instances, AA1 fulfills the new instance set.

Theorems 3, 4 and 5 show that AA1 always fulfills the instance set after any one of the three self-deletion procedures is applied. It follows then from these and the above that AA1 is correct.

Remark:

A node's output cannot be inverted in AA1; however, since only the AND and OR functions are used, Boolean logic guarantees that an equivalent result can be obtained by inverting the node's inputs and changing the node's function from OR to AND, or vice versa. The entries in the NT can then simply be inverted (note that ? is its own inverse).

Theorem 3:

Complete discriminant deletion is correct.

Proof:

Let *CN* be a complete discriminant node, other than the top node. The only nodes needed to build up to *CN* are those that are in the directed graph rooted at *CN*. All the other nodes can be removed and the result follows from the above remark, since *CN* can always be made to fulfill the instance set.

Theorem 4:

Non-discriminant deletion is correct.

Proof:

This result follows from the fact that if *NN* is a non-discriminant node, *DN* a discriminant node, and *PN* the parent node resulting from combining *NN* and *DN*, then *PN* always discriminates a subset of *DN* and can thus be replaced by it. We prove this fact for non-inverted inputs. There are three cases.

Case 1: *NN* contains only one value (0 or 1). It is clear then that *PN* is either *NN* or *DN*. So *PN* discriminates a subset of *DN* (i.e., either nothing at all or the same as *DN*).

Case 2: *NN* contains two values ((0 and ?) or (1 and ?)). There are three subcases: a) *PN*'s function is AND and *NN* contains 1s and ?s or *PN*'s function is OR and *NN* contains 0s and ?s, b) *PN*'s function is AND and *NN* contains 0s and ?s, c) *PN*'s function is OR and *NN* contains 1s and ?s. In case a), *PN* is *DN* with some 0s or 1s possibly replaced by ?, and thus *PN* clearly discriminates a subset of *DN*. In case b), *PN* is *NN* with some ?s possibly replaced by 0, hence *PN* contains only 0s and ?s and is thus a non-discriminant node. So *PN* trivially discriminates a subset (namely the empty one) of *DN*. Case c) is similar to case b), but *PN* now contains only 1s and ?s. In all three possible subcases, *PN* discriminates a subset of *DN*.

Case 3: *NN* contains all three values (0, 1 and ?). Assume that *PN*'s function is the AND function (the case OR is complementary). The 0s and 1s must be in the same column of *NN* and the other column must contain only ?, otherwise we would be in Case 2 (or *NN* would be a discriminant node). Without loss of generality, assume that the N column contains only ?. Then the N column of *PN* will have 0s wherever *DN* had 0s and ? everywhere else. Let p be a positive cell containing 1 in *DN* and I_p the set of discriminated discordant cells (i.e., cells containing 0 in the N column). The p cell of *PN* contains 0, 1 or ?, and each cell of I_p contains 0. Thus *PN* discriminates a subset of *DN* for p (i.e., the same or nothing). If p contains 0 in *DN*, it also contains 0 in *PN*. However each cell of I_p contains ? in *PN* so *PN* no longer discriminates p. Thus, *PN* discriminates a subset (namely the empty set) of *DN* for *p*. The above applies to any cell *p* of *DN*, hence *PN* discriminates a subset of *DN*.

Theorem 5:

Locally redundant deletion is correct.

Proof:

Both cases of locally redundant deletion applicable to AA1 are essentially special cases of non-discriminant deletion. Therefore, this theorem follows almost immediately from Theorem 4.

4. Discussion

4.1. Inconsistency

One important aspect of real-world training sets is their inherent inconsistency. There are at least two sources of inconsistency in a training set. The first is due to the fact that in real-world situations the training set is obtained from experimentation and thus may contain errors. The second one has to do with the fact that certain applications may not be characterized as functions, but rather as distributions (e.g., the task of classifying objects into overlapping classes).

A very important ability, inexistent in most training set learners, but found in AA1, is the ability to use general rules (short-hand instances) in learning. This allows one to construct an instance set that reflects human learning by first presenting general rules to the system, followed by exceptions. It has long been a criticism of neural networks, that they can not incorporate general rules, and thus are very limited models of human learning. AA1 handles both very specific examples and very general rules. A side-effect of such ability, however, is the introduction of inconsistencies (in the form of exceptions) in the instance set.

In AA1, inconsistency is solved by giving precedence to the newer instances. This simple scheme, though reasonable, has several disadvantages. It requires a smart teacher that knows something about the target function and can thus give the instances in the correct order. Also, it is not necessarily best to try to maintain the instance set consistent (nature is replete with inconsistencies). Current research seeks to expand the learning scheme by removing AA1's rigid precedence given to newer rules and by allowing inconsistencies to subsist (see for example [1]).

4.2. Complexity

The hierarchical architecture of AA1 guarantees that execution is O(log*n*) where *n* is the number of nodes in the network. During learning, note that only the training instance requires full execution of the learning algorithm. Modified instances in the add-list are still fulfilled and will therefore not cause node selection, addition and combination. Learning is bounded by a polynomial function of the number of instances in the instance set.

4.3. Generalization

In Section 3, we proved that AA1 converges on any arbitrary Boolean instance set. A more interesting aspect of AA1 has to do with generalization, or its ability to induce new knowledge from past experience. The instance set consists of only a subset of the complete function to be learned, and AA1 must guess the remainder of the function. Even though there is no guarantee that the final network is smallest, AA1's bias towards simplicity results not only in increased parsimony, but also in the ability to generalize. This bias is manifest in two ways. First, for each new instance, the output of the current network is checked and when found concordant no changes are made to the network. Second, after a new instance has been processed, unnecessary nodes are deleted. Empirical studies confirm these findings.

The originality of AA1 lies in the fact that it does not generalize on the basis of proximity of the new input to one of the learned instances, but rather on the basis of how well the network is able to discriminate the new input from learned features of the opposite class.

4.4. Extensions

Several extensions can be made to AA1. For example, the greedy search used in Node Selection, as well as the choice of inputs to be used to create new nodes in Node Addition, could be enhanced by heuristics. Another important problem to be addressed is AA1's memory requirement at each node. It is clear that each node need not store a D column in its node table. A global D column would be sufficient to meet the goal of Node Selection. However, it is still unclear how to do away with (or effectively replace) the P and N columns.

5. Simulation Results

AA1 was tested on several real-world applications, drawn from the Irvine machine learning database [8]. No minimization was attempted, and self-deletion was limited to complete discriminant deletion. The results are summarized in Table 1. For Congressional Voting Data and Hepatitis, the results are averages (on the test set) over 10 runs of AA1. In each run, the instance set and the test set were regenerated with 9/10 of the data used for training and 1/10 used for testing.

Table 1 - AA1 Simulation Results

Congressional Voting Data	92.4%
Monk1	98.6%
Monk2	82.6%
Monk3 (with noise)	91.2%
Monk3 (without noise)	99.1%
Hepatitis	70.3%

6. Summary

This paper presents a formal review of Adaptive Algorithm 1 (AA1) of Adaptive Self-Organizing Concurrent Systems (ASOCS). After a brief overview of AA1 and a description of the algorithm, a detailed proof of correctness for AA1 is given. A series of lemmas and theorems that capture the actions of the algorithm are stated and formally proved, thus showing that AA1 is guaranteed to converge on any arbitrary Boolean instance set. The question of inconsistency and AA1's solution to it are discussed. Intuitive reasons for AA1's ability to generalize are briefly outlined. Extensions to AA1 are suggested. Results of simulations of AA1 on real-world data sets are also reported and show that AA1 has promising generalization performance.

7. Limitations

Despite its many strengths, including the ability to generalize, the combination of rules with examples, the use of self-organization, and low-order polynomial complexity, AA1 has several important limitations. Some of them are discussed here.

A side-effect of the ability to handle both rules and examples is the introduction of inconsistencies (in the form of exceptions) in the instance set. AA1 imposes consistency on the knowledge base by giving precedence to the newer instances, and through various *ad hoc* mechanisms. This tends to reduce the flexibility of the system and seems to run contrary to the way human knowledge is acquired. The precedence rule requires a smart teacher that knows something about the target function and can thus give the instances in the correct order. Moreover, it is not necessarily best to try to maintain the instance set consistent (nature is replete with inconsistencies).

AA1's outcome is order-dependent, that is, the outcome of learning depends upon the order in which instances (rules or examples) are presented to the system. Essentially, the existence of a smart teacher, responsible for the *a priori* ordering of all instances, is assumed. Though realistic in certain limited domains, this assumption is generally not reasonable and the system should be robust enough so as to produce appropriate results in less than optimal ordering cases.

Finally, AA1 is limited to Boolean attributes. Again, the system's flexibility and applicability are hindered. Reasoning power is limited to classification.

CHAPTER 3
Using Precepts to Augment Training Set Learning*

This chapter presents work based on my Master's thesis [GIR93a]. It introduces the concept of precept-driven learning, and corresponds to our first attempt at overcoming some of AA1's limitations, and designing an order-independent, incremental learning system, that combines inductive learning with prior knowledge.

The notion of precept generalizes AA1's rules and the inherent symmetry in PDLA reduces the effect of ordering on the outcome. PDLA's generalization scheme is also more straight forward than that of AA1 for comparable results. The original conclusion of the paper has been extended slightly.

1. Introduction

Machine learning seeks to devise mechanisms that machines can use to learn, rather than being explicitly programmed for, real-world applications. If A is an application having inputs $i_1,...,i_n$ and outputs $o_1,...,o_p$, with domains $I_1,...,I_n$, $O_1,...,O_p$, respectively, then $I=I_1\times...\times I_n$ is the input space and $O=O_1\times...\times O_p$ is the output space for A. In this work, A is a function from I to O.

A common approach to machine learning, called *training set learning* (*TSL*) consists of approximating A from a subset of $I\times O$, called the training set. The training set is commonly derived from experimentation. TSL has become popular for both practical and philosophical reasons. It is practical since, for any application A, it is usually possible to come up with a rich training set for A. It is also philosophically attractive because a training set learner extracts the critical features of A from the training set, without human intervention. These critical features are used in turn to approximate A. Empirical studies show that good results can be achieved with TSL [MOO89, SEJ87].

However, TSL has several drawbacks. Training set learners (e.g., backpropagation) are typically slow as they may require many passes over the training set. Also, there is no guarantee that, given an arbitrary training set, the system will find enough good critical features to get a reasonable approximation of A. Moreover, the number of features to be searched is exponential in the number of inputs, and TSL becomes computationally expensive [BLU87]. Finally, the scarcity of interesting positive theoretical results suggests the difficulty of learning without sufficient a priori knowledge.

This paper presents *precept-driven learning* (*PDL*). PDL is intended to overcome some of TSL's weaknesses. In PDL, the training set is augmented by a small set of *precepts*. A pair $p = (i, o)$ in $I\times O$ is called an *example*. A precept is an example in which some of the *i*-entries (inputs) are set to the special value *don't-care*. An input whose

---

value is not don't-care is said to be *asserted*. If i_j is an input whose value is don't-care in some pair p, then every pair p' obtained from p by asserting i_j to any one of its possible values (in I_j) and leaving the other inputs of p unchanged has the same output as p. Informally, this says that in the context of p, the value of i_j has no effect on the value of the output. The use of the special value don't-care is therefore as a short-hand. A pair containing don't-care inputs *represents* as many examples as the product of the sizes of the input domains of its don't-care inputs.

An example only represents itself (all of its inputs are asserted). Thus, it corresponds to a very specific state of the world. A precept is more general since it represents several examples. A precept reflects some high-level-knowledge about the real-world. It "suggests something advisory and not obligatory communicated typically through teaching" (Webster's Dictionary). Precepts are identical to critical features except that they are given by the teacher and not derived by the algorithm. Also, they are not necessarily correct, but convey some hint or bias. For instance, when dealing with how weather conditions affect what one wears, a possible precept is: "if it rains, take an umbrella," regardless of the temperature, the day of the week, or any other unnecessary (don't-care) detail.

The majority of learning algorithms do training set learning. Several models exist that allow the presentation of precepts such as ASOCS [MAR91a, MAR91b, MAR93] and ScNets [HAL90]. PDL seeks to overcome ASOCS rigid order-dependency in dealing with inconsistency and to improve generalization.

PDL is psychologically plausible. Seldom do humans learn exclusively from random examples, without ever being exposed to more general (common-sense) rules. For example, regardless of most external circumstances, it does not make sense to put your hand on a red-hot stove. This is a general rule or precept that does not require successive examples of settings involving your hand on the stove, to be acquired. Rather, humans are exposed to precepts taught by others (e.g., "it will burn you"), which, together with a built-in pain mechanism, allow the (almost instantaneous) learning of the more general rule. Foreign-language students first learn English rules (e.g., the plural of a noun is formed by tagging an *s* to the end of the singular form of that noun), and are then exposed to exceptions (e.g., geese instead of gooses). Moreover, PDL speeds up learning by pruning the input space and improves generalization by using precepts.

Section 2 gives the details of a precept-driven learning algorithm (PDLA) capable of handling both correct and incorrect precepts. The approach used follows justice's assertion that individuals are innocent until proven guilty. It consists of always assuming that precepts are correct until enough evidence may be obtained from the training set to decide otherwise. Section 3 describes and analyzes the results of simulations of PDLA against several data sets. Performance is measured: 1) without precepts (pure training set learning), 2) with correct precepts, and 3) with incorrect precepts (contradicting the features of the application). Section 4 concludes the paper.

2. A Precept-Driven Learning Algorithm

In this section, we describe a precept-driven learning algorithm (PDLA). Before going into the details of it, we give a few preliminary definitions illustrated by examples. Pairs refer to either examples or precepts.

2.1. Definitions

- Two pairs are *equal* if their input values are equal.
- A pair *p* is a *superset* of a pair *q* if *p* represents all of the examples that *q* represents, and more. If *p* is a superset of *q*, then *q* is a *subset* of *p*.
- Two pairs are *discriminated* if they differ in the value of at least one asserted input. If they differ exactly on one asserted input, then there is a *near_match* between them.
- Pairs that are not equal, and do not satisfy the subset, superset or discriminated relations, are said to *overlap*.
- Two pairs are said to be *concordant* if their outputs are equal; otherwise, they are *discordant*.
- A pair *p contradicts* a pair *q* if *p* and *q* are discordant and *p* is subset or overlap of *q*.
- *size*: the number of examples the pair represents.
- *num_asserted*: the number of asserted inputs.
- *specificity*: the ratio num_asserted/number of inputs. Examples have specificity=1 (maximally specific), while precepts have specificity<1 (more general).

A pair can be *generalized* in many different ways (see [MIC83a] for a list of generalization rules). However, the result is always the same. If a pair *p* represents a set *S* of examples, then a pair *p′* that generalizes *p* (if such a generalization exists) represents a set *S′* of examples such that $S \subseteq S'$. Hence, specificity(*p′*)≤specificity(*p*).

PDLA implements a network of simple nodes. Each node stores a pair *p* and a *list* of the pairs that contradict *p*. Nodes have the following attributes:

- *num_matched*: the number of the stored pair's asserted inputs that match the network inputs.
- *activation*: the node's activation defined as *num_ matched* / *num_asserted*.
- *num_exceptions*: the number of examples that contradict the stored pair.

The network architecture is that of a binary tree. The only requirement is for a broadcast and gather facility. Both operations are O(log*n*) where *n* is the number of nodes in the tree. The tree starts empty and dynamically adapts to new knowledge. During learning, nodes are added and deleted. When a node is added, its position is determined so as to maximize breadth. When a node is deleted, transformations are applied to keep the tree balanced. Hence, the tree is maintained as full as possible. For details on such maintenance mechanisms, see [RUD91].

2.2. Example

Let p1=(<0,0,*,1,*>,<0>), p2=(<0,0,0,1,0>,<1>), p3= (<0,*,*,1,1>,<1>), p4=(<0,0,1, 1,1>,<0>). All variables are Boolean. The following illustrate the definitions:

- p1 and p3 are precepts, p2 and p4 are examples.
- p2 and p3 are concordant, p1 and p2 are discordant.
- size(p1) = size(p3) = 4, size(p2) = size(p4) = 1.
- p1 is a superset of p2, p4 is a subset of p3, p1 and p3 overlap, p2 and p4 are discriminated, there is a near_ match between p2 and p3.
- p2 contradicts p1, p2 and p4 do not contradict each other since they are discriminated.
- num_asserted(p2) = 5, num_asserted(p3) = 3.
- specificity(p1) = .6, specificity(p4) = 1.

Suppose the current network has three nodes storing p1, p2 and p3, respectively, as shown in Figure 1.

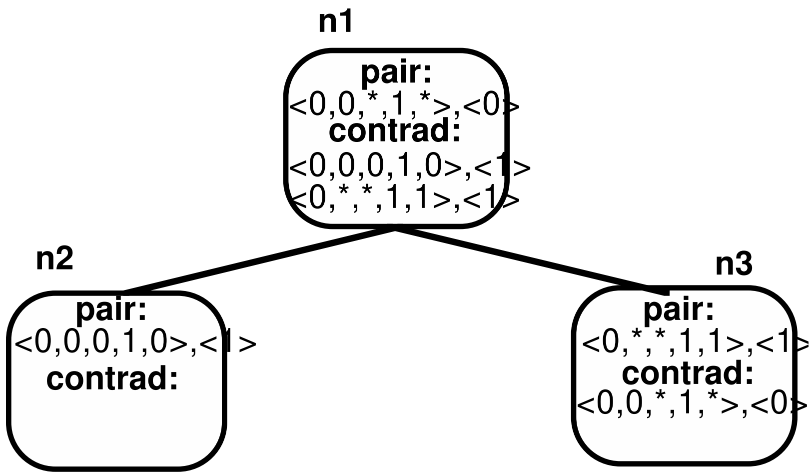


Figure 1 - A Sample Network

Then, for example, num_exceptions(n1)=3, and num_ exceptions(n3)=2. When the input is <0,0,1,1,0>, num_ matched(n1)=3, activation(n1)=1, and activation(n2)=.8.

2.3. PDLA Overview

PDLA can run in two modes: learning and execution. During learning, training pairs are broadcast to all the nodes, and each node reacts to the new pair. It may:

- self-delete,
- update its list of contradictions,
- generalize or cause the new pair to do so,
- attempt to prevent the creation of a new node.

Each node's action is determined by the relation of its stored pair to the training pair. A new node is created if the training pair generalizes or if there does not seem to be a "natural" way for the current network to predict its output. All existing nodes that can generalize do so. If the training pair can generalize and if more than one node causes it to do so, only one generalization is chosen at random, and the node responsible for it self-deletes. Once all training pairs have been presented, exceptions are processed and invalid precepts are deleted.

During execution, the network's inputs are set and each node computes its activation which is a measure of how closely its stored pair matches the network's inputs. The node with highest activation wins and the output part of its stored pair becomes the output of the network. In case of a tie, priority is first given to the most specific pair and then to the one that has the least number of exceptions.

2.4. The Learning Algorithm

We present PDLA and immediately show an example which explains the details of the algorithm as well as the intuition involved in its design. The network starts empty. Training pairs are presented incrementally and learning proceeds as follows. The *create_flag* keeps track of whether a new node must be created for the training pair and the *modified_flag* keeps track of whether the training pair is candidate for generalization.

```
Begin PDLA
  For each training pair q
    Create node nn for q with list of contradictions = ∅
    Create_flag = true and modified_flag = false
    For all nodes n (storing pair p) in the network
      Case superset(n.p, q)
        If concordant(n.p,q) Create_flag = false
        Else Update_List_Contradictions(n)
      Case equal(n.p, q)
        If concordant(n.p,q) Create_flag = false
        Else n self-deletes
      Case subset(n.p, q)
        If concordant(n.p,q) n self-deletes
        Else Update_List_Contradictions(nn)
      Case overlap(n.p, q)
        If discordant(n.p,q)
          Update_List_Contradictions(nn)
          Update_List_Contradictions(n)
      Case discriminated(n.p, q)
        Diff = |num_asserted(n.p)-num_asserted(q)|
        If concordant(n.p,q)&near_match(n.p,q)&diff≤1
          If size(n.p)≥size(q)&num_asserted(n.p)>1
            Drop variable in n.p, Create_flag = false
          Else if num_asserted(q)>1
            Modified_flag = true
    EndForAll          /* all nodes */
    If modified_flag
      Randomly select one of the nodes that set it
      Replace q by its corresponding generalization (q′)
      Delete the selected node, Retain nn (with q′)
    Else if create_flag retain nn
    Else delete nn
```

EndForEach /* each training pair */
Process_Exceptions()
End PDLA

Note that the algorithm automatically handles precepts. All possible relations between all existing nodes and the training pair are checked. Modifications due to dropping variables and updating exceptions are made both to the existing nodes and the new pair. It follows that precepts can be entered in at any time. However, since PDLA is incremental, the order of presentation may affect the outcome. Showing precepts first may speed up learning.

Note also the inherent parallelism of the "For all" loop. All the nodes can execute that loop in parallel.

2.5. Detailed Example of PDLA

Consider the following Boolean training set:

p1 = (<0,0,∗,1,∗>,<0>)
p2 = (<0,0,0,1,0>,<1>)
p3 = (<0,0,1,1,1>,<0>)
p4 = (<1,0,0,1,1>,<1>)
p5 = (<1,0,0,1,1>,<0>)
p6 = (<1,1,0,1,1>,<0>)
p7 = (<0,∗,∗,1,1>,<1>)

Assume that the pairs are presented to the system in the order p1 through p7. The network starts empty.

Presentation of p1 (<0,0,∗,1,∗>,<0>). A new node, say n1, is created for p1 and since the network is empty, it is retained. The resulting network is in Figure 2.

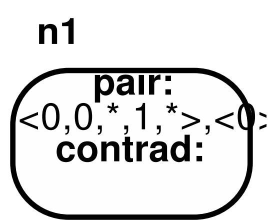


Figure 2 - Network After Presentation of p1

Presentation of p2 (<0,0,0,1,0>,<1>). A new node is created for p2. Node n1 executes the "For all" loop. Its stored pair is superset, discordant. Update_List_Contradictions(n1) executes, causing the addition of p2 to the list of pairs that contradict p1 in n1. The new node, call it n2, is retained. The resulting network is in Figure 3.

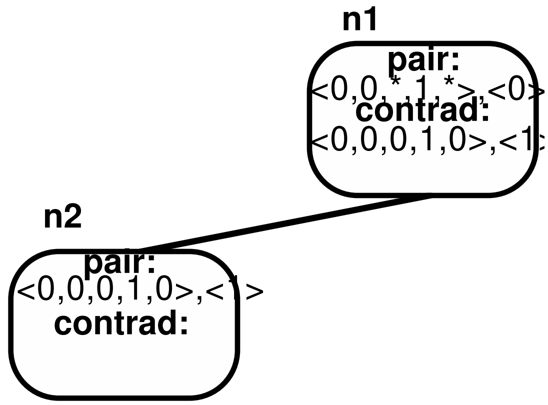


Figure 3 - Network After Presentation of p2

To handle them, PDLA must keep track of exceptions (function Update_List_ Contradictions). Each node stores a list of all the pairs that contradict, or are exceptions to, its stored pair. An important issue must be addressed however, namely that of redundancy. If a pair *p* has been contradicted by a pair *q* and is later contradicted by a pair *q′* that is either a subset of, superset of or equal to *q*, then certain precautions must be taken. If *q′* is equal to *q*, discard *q′* (identical exceptions count only once). If *q′* is a subset of *q*, discard *q′* since *q* accounts for the exceptions due to *q′*. If *q′* is a superset of *q*, replace *q* by *q′* (dual of the previous case). Then, num_exceptions is the sum of the amounts of overlap of input space between each pair in the list and the stored pair. The function Process_ Exceptions handles exceptions and is discussed below.

Keeping a list at each node is expensive in terms of storage (O(*n*) where *n* is the size of the training set), and updating the list is computationally expensive (linear in the size of that list). However, unless precepts are really bad (i.e., inconsistent with the training set), the list of contradictions is expected to be small. Moreover, nodes storing such bad precepts are removed from the network.

<u>Presentation of p3 (<0,0,1,1,1>,<0>)</u>. A new node is created for p3. Nodes n1 and n2 execute in parallel. The pair in n1 is superset, concordant, so create_flag is set to false. The pair in n2 is discriminated, discordant, so n2 causes no change. The node created for p3 is deleted and the network is unchanged. It is consistent with intuition not to create a new node for a pair that is a subset of and concordant with another pair already in the network.

<u>Presentation of p4 (<1,0,0,1,1>,<1>)</u>. A new node, say n3, is created for p4. Nodes n1 and n2 execute in parallel. The pair in n1 is discriminated, discordant, so n1 causes no change. The pair in n2 is discriminated, concordant, but there is no near_match, so n2 causes no change. Node n3 is retained. The resulting network is in Figure 4.

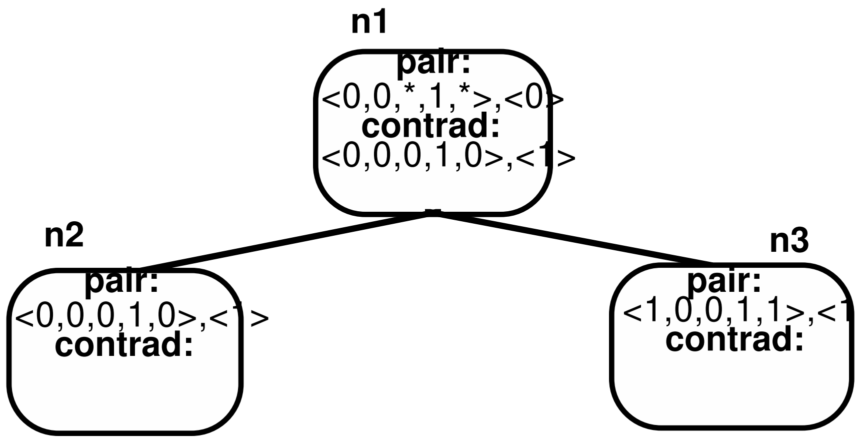


Figure 4 - Network After Presentation of p4

If there does not seem to be a natural way for the current network to predict the output of the training pair, then a new node is created for that pair. In PDLA, natural means that either there exists a node whose pair is concordant, superset, or there is a node causing the new pair to generalize. Neither is true for p4, so n3 is added.

Presentation of p5 (<1,0,0,1,1>,<0>). A new node is created for p5. Nodes n1, n2 and n3 execute in parallel. The pair in n1 is discriminated, concordant, near_match, but *Diff*=2>1, so n1 causes no change. The pair in n2 is discriminated, discordant, so n2 causes no change. The pair in n3 is equal, discordant so n3 self deletes. The node created for p5 is retained. We call it n3 since the current n3 self deletes. The resulting network is in Figure 5.

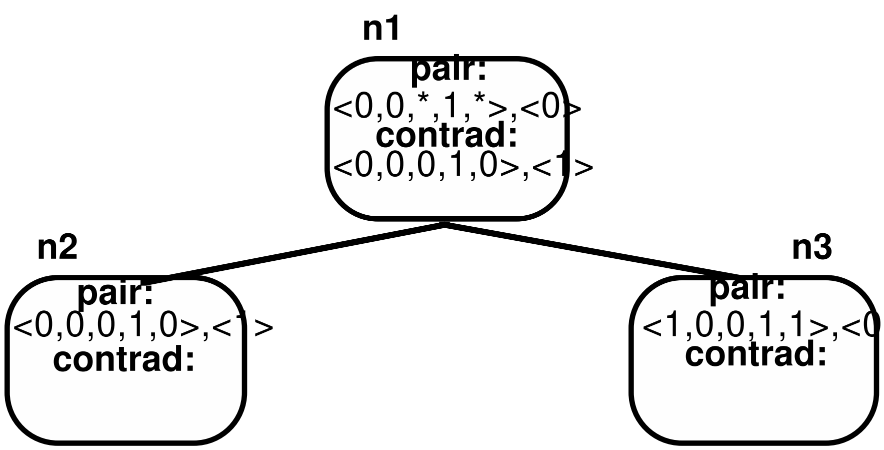


Figure 5 - Network After Presentation of p5

This example shows that PDLA does not handle noise. In this work, *noise* is defined as discordant equal pairs. If two pairs are noisy, only the later one is retained. Another case of noise, which is not addressed here, exists when the application is a distribution rather than a function.

Presentation of p6 (<1,1,0,1,1>,<0>). A new node is created for p6. Nodes n1, n2 and n3 execute in parallel. The pair in n1 is discriminated, concordant, but there is no near_match, so n1 causes no change. The pair in n2 is discriminated, discordant, so n2 causes no change. The pair in n3 is discriminated, concordant, near_match and *Diff*=0≤1 so generalization takes place. The variable on which the two pairs differ is dropped in the pair stored in n3 and create_flag is set to false. The node created for p6 is deleted. The resulting network is in Figure 6.

PDLA uses only the dropping condition generalization rule (see [DIE83]). This rule is applied whenever two pairs are concordant, near_match, and one has 0 or 1 more asserted input than the other (variable *Diff*). The differing input is replaced by the value don't-care in the pair that has the largest size, as long as num_asserted>1. The condition on relative sizes maximizes generalization. The condition on num_asserted ensures that no rule is created whose inputs are all don't-care and that therefore represents all of the input space. *Diff* measures the "distance" between two pairs and is used to control the level of generalization.

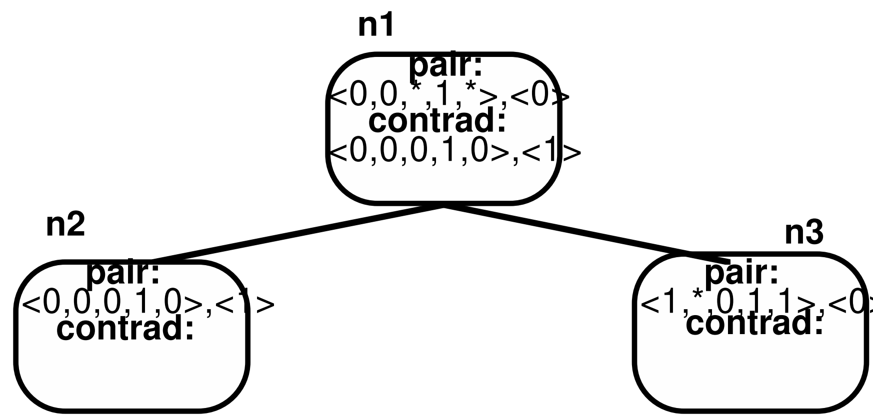


Figure 6 - Network After Presentation of p6

Let *p* and *q* be concordant, Boolean, near_match pairs. *Diff*=0 does not result in generalization, only a smaller network. With *Diff*=*n*, generalization results in a pair that is a superset of both *p* and *q* as well as of at least 2^n-1 unseen pairs. For *n*>1, this tends to give very strong generalization (worse with variables other than Boolean). One could argue that incorrect generalization would be dealt with via exception handling. Though this is true, managing the list of exceptions is expensive. The value 1 was empirically found to give the best results.

All existing pairs that can, generalize, thus maximizing overall generalization. If the training pair can generalize, only one variable is dropped, to facilitate implementation.

<u>Presentation of p7 (<0,*,*,1,1>,<1>)</u>. A new node, n4, is created for p7. Nodes n1, n2 and n3 execute in parallel. The pair in n1 is overlap, discordant, so Update_List_ Contradictions applies to n1 and n4. The pair in n2 is discriminated, concordant, near_match, but *Diff*=2>1, so n2 causes no change. The pair in n3 is discriminated, discordant, so n3 causes no change. Node n4 is retained. The resulting network is in Figure 7.

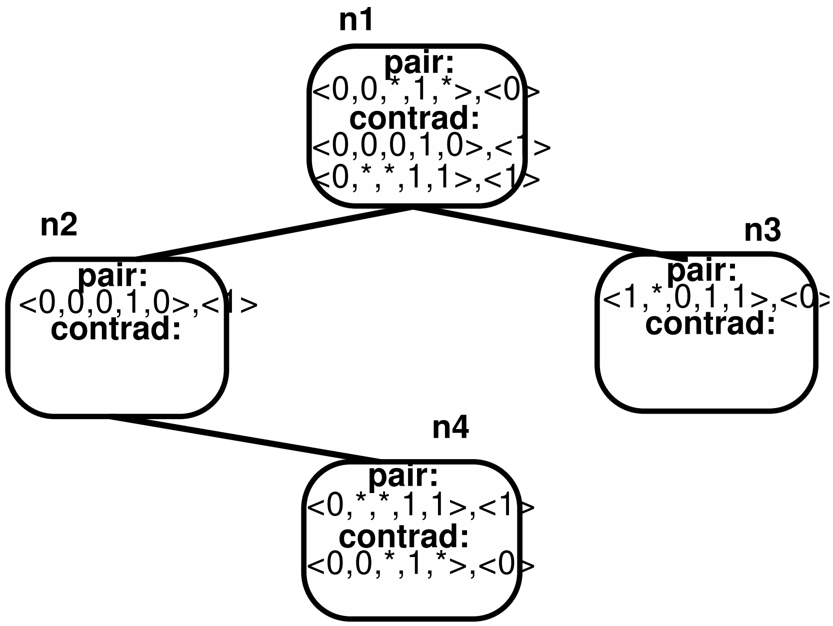


Figure 7 - Network After Presentation of p7

Process_Exceptions(). After all training pairs are seen, exceptions are handled. Precepts are always assumed to be correct until enough evidence is found to refute them. Enough evidence is: num_exceptions(*n*) > 1/2 size(*n.p*). The threshold of half seems reasonable, and is confirmed by empirical results. A smaller threshold may cause the deletion of general rules in favor of their exceptions, and a larger threshold may not delete enough incorrect rules.

Here, n1 is deleted since num_exceptions(n1)=3>2= size(n1.*p*)/2; n4 is not deleted since num_exceptions(n4) =2≤size(n4.*p*)/2. The final network is in Figure 8.

2.6. The Execution Mode

During execution, the network is presented unknown inputs and must guess the correct output. Each node computes its activation, and a winner is found as follows:

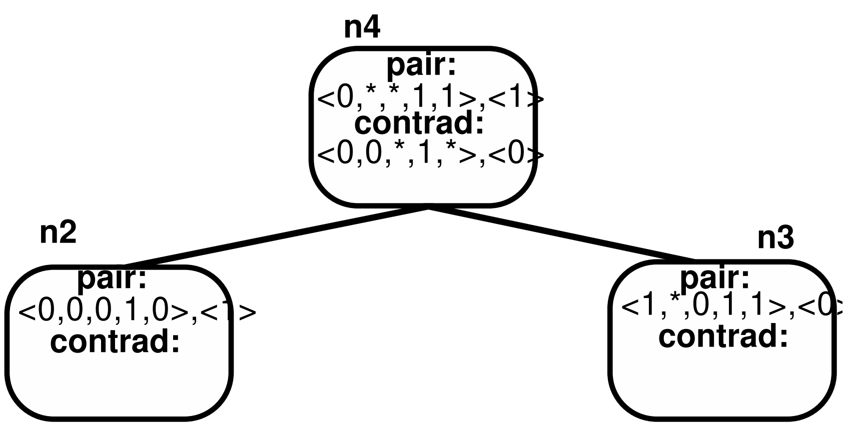


Figure 8 - Final Network

Begin Exec
  If there is a single node with highest activation, it wins
  Else if two nodes *n* and *m* are tied
    If specificity(*n.p*)>specificity(*m.p*) *n* wins
    Else if specificity(*n.p*)=specificity(*m.p*)
      If num_exceptions(*n*)<num_exceptions(*m*) *n* wins
      Else *m* wins
    Else *m* wins
  Network output = Output of pair in winning node
End Exec

Recall that activation measures closeness of match. When two nodes are tied for highest activation, priority is given to the one whose stored pair is more specific. In particular, if a new pair is a subset of a general rule, but yet is equal to a more specific pair, then it is one of the rule's exceptions and should be recognized as such. If the new pair is not equal to one of the seen exceptions, then its output will be determined by the general rule. This priority scheme makes it possible to deal with exceptions, but also has the tendency to learn noise.

When two nodes are tied and have the same specificity, priority is given to the one with fewer exceptions, thus the output is set to the value that is most likely to be correct.

3. Simulation Results

PDLA was tested on several applications, drawn from the Irvine machine learning database [MUR92], with two modifications. All real-valued attributes are made discrete (using a K-means algorithm on each input), and all don't-know values are treated as don't-care values.

For each data set, results are reported with and without precepts. The number and generality of the precepts vary. Precepts were obtained from the author's common sense knowledge about the application's domain (Lenses, Zoo, Hepatitis) or from a priori knowledge about the problems (Iris, Voting, Credit-Screening, Chess). Tests were run with both correct and incorrect precepts. The incorrect precepts were obtained from the correct ones by changing the value of their outputs.

Each of the reported results is an average over 10 runs of PDLA. In each run, the training set and test set are randomly regenerated. Precepts remain unchanged and are presented first. Both training set and test set contain one half of the examples in the complete data set. Results are summarized in Table 1. The numbers following the percentage of correct guesses, or predictive accuracy (PA) correspond respectively to training set size (TS), including precepts where applicable, and final network size (NS) after the learning phase (in number of nodes).

Table 1 shows that the addition of simple precepts improves generalization performance and significantly reduces the size of the network (greater parsimony). On average, PA is increased by 4.3%, while NS is reduced by 30%. Table 1 also shows that PDLA has some ability to deal with bad precepts. In most cases, the "recovery" is good, but is obtained at the expense of a larger network.

Table 1 - PDLA-Simulation Results

	No precepts			**Good precepts**			**Bad precepts**		
Application	PA	TS	NS	PA	TS	NS	PA	TS	NS
lenses	67.5	12	6	78.3	13	4	50.8	13	6
zoo	95.8	45	22	95.8	46	19	85.1	46	23
voting	88.5	217	73	91.0	218	31	82.2	218	72
iris	92	75	33	92.7	81	22	68	81	38
hepatitis	77.1	77	71	79.5	79	61	61.4	79	72
credit-scr.	79.4	345	343	83.3	346	203	50.5	346	304
chess-ends	90.5	1598	937	92.5	1599	770	90.5	1599	937
mushroom	100	4062	1499	(unnecessary)			(unnecessary)		

4. Conclusion

This paper introduces precept-driven learning and describes a precept-driven learning algorithm (PDLA). Results of simulations on several data sets are reported. PDLA's main features include:

- Distributed implementation facilitating the use of massive parallelism and self adaptation,
- Bounded learning and execution times complexity,
- Use of precepts resulting in faster learning, increased parsimony, and improved generalization,
- Ability to deal with incorrect precepts.

Though PDLA overcomes some of AA1's limitations, it also suffers some important drawbacks. Reasoning power is still limited to classification. Attributes may be nominal, but cannot be continuous (or even ordered). Recent efforts at extending PDLA include rebroadcasting modified pairs, making the system more inductive by only processing pairs that are not correctly classified, and replacing bad precepts by new ones (e.g., toggle output). Future research should focus on noise handling, improved generalization schemes, dealing with missing inputs, and alternatives in handling exceptions.

CHAPTER 4
Seven Desirable Properties for Artificial Learning Systems*

In this chapter we discuss some of the basic philosophy underlying the dissertation. In particular, we present a set of characteristics that are believed to be useful to the design of artificial systems that combine learning with prior knowledge and reasoning. Section 2 is not in the published paper, and a section reviewing PDL2 as proof of concept is omitted. All of the algorithms presented in this dissertation serve as proof of concept, by attempting to integrate the above properties into a unified framework in which inductive learning is combined with prior knowledge and reasoning to yield models capable of dealing with a greater variety of interesting problems.

1. Introduction

Over the last 30 years, extensive work has been done in the area of Machine Learning. Whether the approach is inspired by biological plausibility (Artificial Neural Networks) or by psychological plausibility (Artificial Intelligence), the goal is to design systems capable of learning and reasoning about certain tasks (e.g., analogy, classification, etc.). Most existing systems focus on some observed aspect of human learning, and attempt to match, and possibly exceed, human abilities. For example, several inductive models exist that exhibit similar (and sometimes better) predictive accuracy than their human counterparts on various problems (e.g., loan underwriting, mine-rock discrimination). It is a system's ability to capture and exhibit certain important characteristics (often inspired by human learning), that allows it to be useful as an artificial learning system.

We propose seven desirable properties for artificial learning systems, namely: incrementality, non-monotonicity, inconsistency and conflicting defaults handling, abstraction, self-organization, generalization, and computational tractability. This proposed set of properties is not claimed to be complete, nor does it imply that all properties must necessarily be present in all learning models. Rather, it focuses on issues that are not often explicitly addressed and provides a basic foundation for the design of more efficient algorithms. As most other computer-based systems, learning systems are difficult to reverse engineer, and desirable properties should be part of the original design rather than retrofitted into the system after the fact.

In the following sections, we examine each property in turn and show how their use can improve learning and reasoning, and potentially widen the range of applications of artificial learning systems.

2. Why Should Machines Learn?

---

* In *Proceedings of the Seventh Florida AI Research Symposium (FLAIRS'94)*, 16-20. Copyright retained.

Before we embark in a discussion of properties of artificial learning systems, we briefly summarize some arguments for why machines should indeed learn. Our motivation is practical, and we make no claim as to what is possible or not possible with Artificial Intelligence. For insightful discussions of such (and related) matters, see [DRE79, DRE92, RIB75, RIN79]. We only list our main arguments here.

- Some algorithms are hard to develop. In particular, not everything may be computable in the usual sense.
- Machines are fast. Once they have learned a task, they may become more efficient than their human counterparts.
- Machines are not as vulnerable as humans. In particular, they have no feelings and are not subject to most human hazards (disease, radioactivity, etc.).
- Learning is more robust than programming. In particular, no redesign or recoding is necessary. Only new information need be given and the machine adapts (much as humans do).
- Learning is more adaptive than programming. In particular, only data need be given. With unsupervised learning the machine can even discover classifications on its own.
- Constructing artificial learning systems provides us with much insight into the process of learning. By better understanding how learning takes place, we may become better teachers as well.

3. Incrementality

Humans acquire knowledge *incrementally*, that is, they learn over time. All the information necessary to learn a given concept is rarely available a priori. Rather, new pieces of information become available over time, and the knowledge base is constantly revised (i.e., evolves) based on the newly acquired information. To have all of the knowledge a priori amounts to programming rather than learning. One would then have to assume the existence of a "smart" teacher that can program the system to perform the proposed task. Unless the teacher is another machine, such an assumption seems to go against the long term goal of building artificial systems capable of learning. Moreover, in the context of learning grammar with a recurrent network, Elman [ELM91] also gives empirical evidence that the "network fails to learn the task when the entire data set is presented all at once, but succeeds when the data are presented incrementally."

Another aspect of incrementality has to do with the form in which knowledge is acquired. Because of built-in mechanisms (e.g., pain) and social structures (e.g., the family, school), humans are able to combine both specific examples and more general rules to efficiently learn complex problems. For example, as shown in [GIR93a], it is common that humans first get exposed to general rules and then to exceptions. Elman [ELM91] suggests that the same form of incrementality may actually be the result of an early handicap (e.g., limited memory and attention span characteristic of children). In other words, it may be that the learning system itself evolves over time, rather than the knowledge it is presented. In the context of Elman's experiment, complete English sentences are always presented. However, during early stages of learning the system uses only subparts (e.g., noun-verb combinations) of the sentences to induce general and unrefined grammar rules. Then, during latter stages of learning the system assimilates the complete sentences and can thus refine these rules.

In either case, the ability to use rules and examples increases learning speed by pruning and constraining the search in the input space, reduces memory requirements, and improves overall predictive accuracy (see, for example [GIR93b]). Moreover, with appropriate generalization mechanisms, examples can effectively supplement rules given a priori by generating new rules, and modifying existing ones. Hence, the system combines the intensional approach (based on features, expressed by rules) and the extensional approach (based on instances, expressed by examples). Such a combination may prove useful in commonsense reasoning. In particular, it allows the system to retain the self-adaptivity inherent in human learning, while not having to unnecessarily suffer from poor or atypical learning environments.

Chronology is inherent in incrementality. Thus it is also possible to implicitly integrate time as a factor and another source of bias in learning. For example, the system could choose to give precedence to newly acquired knowledge, recognizing the possibility that early guesses may be incorrect (see, for example [MAR94]). In an iterated training set system, the distribution of "weight" over the various pieces of knowledge is uniform. So, a temporal ordering of the various pieces of knowledge would have to be made explicit (most likely by some external teacher) to achieve the same result.

Note also that if only minor revisions are made to the current knowledge base as new information is gained, and no major reprocessing of previous observations need be done, then an incremental system may allow a reduction of overall algorithmic complexity.

4. Non-Monotonicity

Classical first-order logic is monotonic, that is, if A is a set of axioms and C is derivable from A, then C is also derivable from any superset of A. Informally, this says that the acquisition of new knowledge can never invalidate previously known facts. Though theoretically convenient, monotonicity certainly does not hold in everyday life, where information is often uncertain and incomplete. Consider the following classical example. Suppose someone tells you that Tweety is a bird and asks you whether Tweety can fly. Since you know that birds typically fly, and you have no reason to believe otherwise concerning Tweety, you are most likely to say that Tweety can fly. Now, imagine that later on you are told that Tweety is a penguin. Then, since you know that penguins, though they are birds, do not fly, you will most likely withdraw your previous conclusion and say that Tweety cannot fly. The new piece of knowledge you gained invalidated the truth of the previously accepted fact.

Such patterns of reasoning, often equated with commonsense reasoning by logicians (see, for example [BRE91, LUK90]), are pervasive in the way humans deal with the inherent uncertainty and incompleteness of the world (or their representation thereof). The development of systems that effectively handle classical commonsense protocols, such as inheritance, is a first step in overcoming the brittleness bottleneck.

Non-monotonicity results from incrementality. If all the knowledge is available a priori, then non-monotonicity is not an issue. That is, there is no need for special learning mechanisms that invalidate portions of knowledge, while not affecting the rest of it. Essentially, the world is closed. If knowledge is incrementally made available, however, then more work is required. Consider the above example again. Knowing that Tweety is a penguin should only directly and explicitly affect knowledge about Tweety, not any other birds. Just because you now know that Tweety is a penguin and that penguins do not fly, you (most likely) would not want to conclude that Fred (another bird that I encounter later) does not fly, even though my belief in Fred's flying ability may be (implicitly) lessened by my having discovered the existence of exceptions to the rule.

5. Inconsistency and Conflicting Defaults Handling

Everyday life is replete with inconsistencies and conflicting defaults, such as the famous Nixon Diamond [REI81]. Given that Quakers are typically pacifist, that Republicans are typically not pacifist, and that Nixon is both a Republican and a Quaker, what can you say about Nixon's dispositions? Removing inconsistency does simplify the learning task, but is usually artificial and does not reflect the world. Indeed, as pointed out by Szepesvári and Lörincz [SZE93], internal representations of the world are bound to be partial so that what we are left to deal with are relations rather than functions. Inconsistency is a fact of life.

In the face of inconsistencies, as in the above example, it is not reasonable (as first-order logic would require) to force the knowledge base to become consistent by arbitrarily ridding it of one (or part) of the offending rules. Instead, while both rules are essentially good as default rules, Nixon appears to be an exception to at least one of them. Mechanisms should be developed to handle such situations. If the extensional and intensional approaches are combined, greater flexibility can be achieved. There are at least three available options: 1) default rules are given a priori, along with their respective priority (e.g., religion is stronger than politics), 2) default rules are given a priori, but their relative priorities are computed dynamically via exposition to a set of examples, or 3) default rules and their priorities are inductively learned by the system. The first option guarantees the expected result (according to some teacher), the second option adds flexibility by letting experience supplement a priori knowledge, and the third option produces a model consistent with current experience with the world. These options represent a trade-off between self-adaptivity of the system and consistency of the system with some external entity's representation of the world.

6. Abstraction

Humans are good at dealing with symbols. Many machine learning techniques are limited by their *attribute-value pair* representation language. Attribute-value pairs make it difficult to deal with open domains and higher-level, abstract relationships. For instance, if the relation to be found is equality of two variables (e.g., learning to discriminate between squares and rectangles), then the algorithm may discover a rule for each value appearing in the training set, but may not be able to extend them to all possible values. This suggests that a more symbolic representation be used so that such high-level relationships can be generated when they exist. The need for such a language is likely to be a consequence of the fact that current artificial systems lack the richness of the human sensory system. The more ways there are to observe and interact with an object (visual, tactile, etc.) the richer its final internal representation.

Symbols also tend to be much easier for human interaction and understanding (see, for example, Michalski's comprehensibility bias). There is however a trade-off between the efficiency of numerical representations and the flexibility of more symbolic representations. Neural nets, which use numbers as inputs, are computationally efficient, but they lack a "comprehensible" interface with the human user. This opacity constitutes one of the main criticisms of most numerical approaches [MIN91]. On the other hand, symbol-manipulating systems are easier to interact with and understand, but their efficient implementation remains a computational challenge. Research is currently under way to design representation languages that take advantage of both approaches.

7. Self-Organization

Because the human brain consists of complex networks of neurons, some advocate artificial neural networks as the best approach to machine learning. Though this position is a little extreme (and too committal), neural networks have many interesting features, in particular self-organization and massive parallelism.

Architectures that are massively parallel offer a platform for the design of more efficient algorithms. If knowledge is distributed and simple nodes are used, even mechanisms as simple as *gather and broadcast* can be used to update the entire current knowledge base at once. The use of parallelism can also often aid the problem of computational complexity.

By self-organization we mean the ability to adapt to new information and to consequently dynamically maintain and evolve the knowledge base. The system must be able to learn the given task, rather than the task be explicitly programmed into the system. Self-organization is inherent in human learning since incrementality forces constant adaptation. From a practical standpoint, this means that if artificial systems are self-organizing, then they can potentially be used in environments that present a hazard to humans (e.g., radioactive sites), or in highly interactive settings in which the machine adapts to its user. Appropriate artificial sensory devices must be developed for such applications.

We also suggest, from a more psychological standpoint, that the ability to self-organize constitutes a first step in the direction of self-awareness and metarepresentational ability, which appear to be prerequisites to common sense (see [FOR89]). Indeed, self-organization presupposes that the system knows that it knows something. This awareness may in turn be exploited to represent knowledge about knowledge. In fact, if machines can learn, they could potentially learn how to learn, and consequently teach more effectively.

8. Generalization

The issue of generalization is two-fold. The first aspect is that of the generalization language and what it can generate, and the second aspect is that of predictive accuracy.

The generalization language, which one can equate with the output space of the learning algorithm, determines the kind of concepts that the system can generate. There is often a mismatch between the representation language and the generalization language. Going back to the equality example of Section 4, one can easily see that adding new special symbols to the attribute-value language would allow it to represent equality of two variables. However, the algorithm may still not be able to efficiently generate that relationship (for example, due to the lack of an appropriate generalization rule). Hence, not only the representation language but also the generalization language must be revised. The value of incrementality and self-organization depends in large measure on the ability to generalize.

The other aspect of generalization is predictive accuracy, that is, the ability, based on what one knows or has been trained on, to make correct predictions about new situations. In an iterated training set approach for example, this implies that convergence on the training set is not as important an issue as is predictive accuracy on the test set. Any viable mechanism should exhibit good generalization performance. Incrementality may improve such performance (see, for example [GIR93a]). Predictive accuracy is intimately related to the generalization language, since that language also determines the generality of the generated concepts.

9. Computational Tractability

Artificial learning systems can often be computationally expensive. Many of the current learning models have cast the generalization problem as a search problem (through the space of all possible generalizations) and thus are faced with combinatorial explosion (often aided by heuristics). For artificial models to be of any practical use, they must be computationally tractable. We suggest that the algorithms should have at most polynomial-time complexity. Parallel, self-organizing, incremental implementations can often reduce complexity.

10. Conclusion

The list of properties discussed here is not meant to be exhaustive. We feel that it is very dynamic in nature, and that, as experience is gained, new properties may emerge. We note also that determining membership in such a set of desirable properties is not a trivial task. Consequently, it may be difficult to evaluate whether the set is complete or even useful. Completeness cannot be guaranteed, however, usefulness may be measured in part by empirical studies. For example, does the new property enhance some other aspect of the learning system (e.g., generalization, range of applications, etc.)? Also, psychological and biological evidences in humans may serve as a guide.

Much work still remains to be done to develop learning systems that effectively use the above properties. The algorithms presented in this dissertation all attempt to integrate many of them. Work is ongoing to extend the basic model and develop new algorithms. We are attempting to show that the integration of the properties discussed here can indeed result in significant improvements in learning mechanisms and reasoning performance, and in a widening of the range of applications of artificial learning systems.

CHAPTER 5
An Incremental Learning Model for Commonsense Reasoning*

Upon reading work by Sun [SUN92, SUN93] on a connectionist approach to the representation of and reasoning about commonsense knowledge, and examining PDLA in the new light shed by this work, it became apparent that PDLA could be extended with limited reasoning ability. PDLA already combines inductive learning with prior knowledge. Hence, extending it provides a useful starting place for achieving the goal of this dissertation. This chapter discusses the resulting modifications made to PDLA to handle several commonsense reasoning protocols.

1. Introduction

Much effort has been devoted to understanding learning and reasoning in artificial intelligence [GIN87, SHA90, WEB81]. However, very few models attempt to integrate these two complementary processes. Rather, there is a vast body of research in machine learning, often focusing on inductive learning from examples, quite isolated from the work on reasoning in artificial intelligence. Though these two processes may be different, they are very much interrelated. The ability to reason about a domain of knowledge is often based on rules about that domain, that must be learned somehow. And the ability to reason can often be used to acquire new knowledge, or learn.

This chapter proposes a system that attempts to combine inductive learning from examples with prior knowledge in the form of precepts, and default reasoning. In particular, the system deals with linear and tree-structured inheritance, including exceptions to exceptions. It handles conflicting defaults (such as the famous Nixon Diamond [REI81]) through either static priorities given *a priori* by an external source, or dynamic priorities obtained by counting matching examples in a real-world training set. The system's execution combines rule-based reasoning and similarity-based reasoning, as in [SUN92]. If a new situation can be handled by an existing rule (given *a priori* or learned), it follows the rule, and if there is no matching rule, the new situation follows the most similar (for some similarity measure discussed later) existing situation. Learning is effected incrementally and inductively by retaining those situations that are not accounted for by the current knowledge base. Of interest is the fact that new knowledge may cause generalization of the existing knowledge base.

Indeed, one of the main advantages of the proposed system over similar models, such as [HAL90, SUN92] is its ability to generalize. Rules need not all be given *a priori*. Based on a set of training examples, the system inductively generalizes and synthesizes rules that can then be used in reasoning. Thus, from a psychological standpoint, the model retains the self-adaptive nature inherent in human learning, while allowing the

---

 To appear in *Proceedings of the Seventh International Symposium on Artificial Intelligence (ISAI'94)*, Monterrey, México, October 1994, 1-8 (??). Permission pending.

encoding of *a priori* knowledge. Experience can then be combined with theory to yield a system capable of confirming, refuting, and modifying its "innate" knowledge based on empirical observations.

The combining of inductive learning and *a priori* knowledge in the model also accounts for its ability to deal with several (commonsense) reasoning problems both extensionally and intensionally. Most approaches to date have focused on one or the other. Little (if any) has been done to combine them. This work is an attempt at filling this gap. Results of simulations on several problems demonstrate promise.

Section 2 presents the algorithm PDL2 (Precept-Driven Learning 2) and discusses some of its more original features. Section 3 characterizes PDL2's reasoning scheme and shows how PDL2 overcomes several aspects of the brittleness problem. Section 4 contains preliminary results on several datasets and tasks. Finally, Section 5 concludes the chapter.

2. Precept-Driven Learning

Learning from examples is the process of inductively discovering general rules from exposure to specific examples. This mechanism has been thoroughly studied and several models have been proposed [DIE83, MIC83a, QUI86]. However useful, inductive learning does not take advantage of an important source of knowledge, namely prior knowledge. Human learning is not the sole result of exposure to random examples. Rather, built-in mechanisms (e.g., pain), and social structures (e.g., the family, school) account for much of humans ability to efficiently learn complex problems. Much effort has recently been expended in understanding the sources and use of prior knowledge in learning [DES93].

One way to use prior knowledge is as an initial learning bias, whose purpose is both to prune the input space and to constrain the search for correct generalizations. In this case, the bias can be thought of as a general rule or *precept* resulting from some instantiation of domain knowledge or common sense. It thus "suggests something advisory and not obligatory communicated typically through teaching" (Webster's Dictionary). For instance, when dealing with how weather conditions affect what one wears, a possible precept is: "if it rains, take an umbrella," regardless of the temperature, the day of the week, or any other unnecessary (don't-care) detail. It is our contention that the encoding and use of such precepts improves an inductive learning system's overall efficiency. In particular, it increases learning speed by pruning and constraining the search in the input space, reduces memory requirements, and improves overall predictive accuracy. The precept-driven learning algorithm (PDLA) discussed in [GIR93b] is a proof of concept for the above contention.

The system presented here, called PDL2, is an extension of PDLA. In particular, PDL2 provides mechanisms to handle noise. It also retains exceptions to exceptions in the knowledge base, thus accounting for linear inheritance and cancellation of inheritance. Finally, PDL2 extends PDLA's execution (or reasoning) scheme with both static and dynamic priorities. PDL2 also bears some resemblance to CONSYDERR [SUN92]. CONSYDERR is somewhat more natural in terms of representation language and allows non-uniform weighting of features. However, CONSYDERR does not support learning, and it is not clear how such a skill could be (easily) integrated in the model.

Section 2.1. presents an overview of PDL2. Some of the more original aspects of the algorithm are then discussed. For more details, see [GIR93a].

2.1. PDL2 - Algorithmic Overview

Let A be an application having inputs $i_1, ..., i_n$ and output o, with domains $I_1, ..., I_n, O$, respectively. Then $I=I_1\times...\times I_n$ is the input space and O is the output space for A. Learning A from examples consists of approximating A from a subset of $I\times O$, called the training set. The training set is commonly derived from experimentation. In this work, the domains are symbolic-valued, and the approximation is a function from I to O.

PDL2's representation language is the classical attribute-value language. A pair $p = (i, o)$ in $I\times O$ is called an *example*. A *precept* is an example in which some of the i-entries (inputs) are set to the special value *don't-care*. An input whose value is not don't-care is said to be *asserted*. If i_j is an input whose value is don't-care in a pair p, then every pair p' obtained from p by asserting i_j to any one of its possible values (in I_j) and leaving p's other inputs unchanged has the same output as p. Informally, this says that in the context of p, the value of i_j has no effect on the value of the output. A pair containing don't-care inputs *represents* as many examples as the product of the sizes of the input domains of its don't-care inputs. The number of examples a pair represents is its *size*. If a pair has m of its n inputs asserted, its *generality* is defined by n-m. Precepts are more general than examples.

PDL2 implements a network of simple nodes, each storing a pair, together with the pair's priority value (*pty*), counters for each output value (vector *counts*), a counter for covered pairs (*cover_count*), and an exception flag (*e_flag*). A node's priority is used to handle conflicting defaults (see Section 3.1). A node's counters are used for noise handling. Exactly one counter value is incremented each time an incoming training pair is equal to the node's stored pair. The value incremented corresponds to the training pair's output value. The counter value that is highest represents the "most probable" output for the node's stored pair, and can thus be used as the node's output in execution. A node's cover_count corresponds to the number of training pairs its stored pair covers. We say that a pair p *covers* a pair q if and only if p and q have the same output value, and the set of examples represented by q is a proper subset of the set of examples represented by p. The value of cover_count can be used as a measure of confidence in the node's stored pair. Finally, a node's e_flag is set if its stored pair is an exception to any other pair. We say that a pair p is an *exception* to a pair q if and only if p and q have different output values, and q covers p.

The network architecture is a balanced binary tree. The only requirement is for a broadcast and gather facility. The tree starts empty and dynamically adapts to new knowledge. During learning, nodes are added and deleted. When a node is added, its position is determined so as to maximize breadth. When a node is deleted, transformations are applied to keep the tree balanced. Hence, the tree is always maintained as full as possible (see [RUD91]).

PDL2 runs in two modes: learning and execution/reasoning. Pseudo code for the learning algorithm is given in Appendix A. Learning takes place in two phases. In the first phase, the training pair is broadcast to all the nodes. Each node computes its matching state, which consists of the set relationship between the sets of examples represented by its stored pair and the training pair, and the binary relationship (equal/not equal) between the pairs' outputs. Nodes whose stored pairs are exceptions to the training pair set their exception flags and return false to their parent nodes. Nodes for which the training pair is an exception to their stored pair return true to their parent node. All other nodes return false to their parent nodes. The result is gathered at the root (priority being given to true). If true is returned to the root node, then the training pair is an exception to at least one existing pair.

In the second phase of learning, the value gathered at the root is rebroadcast to all the nodes and each node reacts, based on its matching state. It may:

- self-delete,
- increment priority values, counters, and/or cover counts,
- generalize or cause the new pair to do so,
- attempt to prevent the creation of a new node.

A new node is always created if the training pair is an exception to some other existing pair. A new node is also created if the training pair generalizes or if there does not seem to be a "natural" way for the current network to predict its output.

All existing nodes that can generalize do so. If the training pair can generalize and if more than one node causes it to do so, only one generalization is chosen at random. A newly created generalization is added only if there does not already exist one that is equal to or covers it. The basic mechanism for generalization is the dropping condition rule [MIC83a]. The rule is applied whenever two pairs have the same output value, differ in exactly one asserted input value, and one has 0 or 1 more asserted input than the other. The differing input is replaced by the value don't-care in the pair that has the largest size, as long as it has more than one asserted input. The condition on relative sizes maximizes generalization. The condition on the number of asserted inputs ensures that no rule is created whose inputs are all don't-care and that therefore represents all of the input space. The condition on relative number of asserted inputs controls the level of generalization. Dropping the condition is simple, fast, easily accommodated by the current representation language, and works well with nominal attributes, but it may lead to over-generalization. PDL2's ability to handle exceptions can rectify such over-generalizations. Other mechanisms for generalization (and likely changes to the representation) are the topic of future research.

During execution, the network's inputs are set and each node computes its activation which is a measure of how closely its stored pair matches the network's inputs. If p is the pair stored in node n, then the activation for n is defined to be the ratio of the number of p's asserted inputs matched by the network's inputs to p's number of asserted inputs. This measure is identical to the measure of conceptual similarity of [SUN92]. The node with highest activation wins and the output of its stored pair becomes the output of the network. In case of a tie, priority is first given to the node whose pair is more specific, then to the one that has the highest priority value, and finally to the one whose cover count is highest. This process is easily handled by the broadcast and gather facility.

PDL2's execution mechanism is a combination of rule-based and similarity based reasoning. Indeed, if a new pair matches an existing rule, that rule's activation will be highest (i.e., 1). If there is no matching rule, then various nodes compete and the winner is the most similar. When nodes are tied, giving priority to the more specific one enables the system to recognize exceptions Then, giving priority to the one with highest priority allows the system to deal with conflicting defaults. Finally, giving priority to the one with highest cover count allows the system to generalize to the most likely (based on available evidence) output.

2.2. Uncertainty of Precepts

To properly reflect the uncertain nature of precepts, the system must have mechanisms to deal with both correct and incorrect guesses. As a policy, a precept is always assumed to be correct until enough evidence is available (from experience with specific examples) to the contrary. The question of what constitutes enough evidence is naturally the key to properly handling truly incorrect guesses. The danger is that information might be lost if a precept is mistakenly deleted. In [GIR93b], enough evidence is defined as the existence of examples (or other precepts) contradicting the targeted precept on more than half of the input space it covers. Though reasonable, this scheme often results in loss of information and is inadequate (due in part to the representation language used) for commonsense reasoning (particularly inheritance). Since mechanisms exist to keep track of exceptions and exceptions, because they are more specific, have priority over more general rules, PDL2 does not need to remove precepts.

2.3. Generalization

As was mentioned earlier, the main advantage of PDL2 is that it needs not rely on some external source for its rules. Rather, it may inductively discover those rules from a set of observations. Even with a simple generalization rule, such as the dropping condition rule [MIC83a], the system is able to make a useful inductive leap (see Section 4). The process of generalization allows the discovery of rules that may not have been supplied as precepts (if any were) by an external source, as well as the potential modification of existing rules.

2.4. Incrementality

Assuming that all the knowledge about a certain task is available *a priori* amounts to programming. Though achievable in limited domains, this assumption often does not hold. Humans tend to learn over time. The need for such incremental learning in artificial systems is argued elsewhere [ELM91, GIR94a]. It is inherent to PDL2. Incrementality tends to reduce overall algorithmic complexity and provides a natural mechanism to incorporate chronology as a learning bias (e.g., priority given to newer rules as in [MAR94]).

One further advantage of combining incremental inductive learning from examples and precept encoding is the resulting combination of the extensional (based on examples) and the intensional (based on features) approaches. Indeed, one may encode a precept (along with a static *a priori* priority level) or one may choose to let the system dynamically and self-adaptively discover a precept and its priority (based for example on the relative number of examples satisfying a given condition). Thus, when dealing with birds, one may choose to "hard-code" the fact that typically birds fly, or one may let the system discover the default on its own. On the one hand, the system is guaranteed to conclude that birds usually fly, even if it has seen a majority of penguins so far. On the other hand, the system lacks some self-adaptation and psychological plausibility. Indeed, if all the birds I have seen so far are penguins, my current state of belief is likely to be that birds typically do not fly. Only as I get exposed to more new instances of birds that fly, does my belief evolve, ultimately, and more naturally reaching the "proper" conclusion, that birds other than penguins do fly. Extension allows the system to adapt to and reflect its current environment, while intension allows the system not to unnecessarily suffer from poor or atypical learning environments.

Incrementality presents some challenges however in terms of the learning algorithm. In particular, the order in which learning takes place (e.g., whether rules or exceptions are showed first) may affect the outcome. If enough information is available (i.e., if either the learning period is long enough or the information provided is rich enough), then the system is likely to reach a state consistent with its actual environment, even though it may have been inconsistent at earlier times, as in the above example.

3. PDL2's Reasoning Scheme

As learning takes place, or is completed (i.e., all available examples have been seen), the system becomes available for reasoning. In this section, we characterize PDL2's reasoning scheme, discuss how PDL2 overcomes many aspects of brittleness, and show how PDL2 handles the important commonsense problem of inheritance. We follow the characterization of the brittleness problem given in [SUN92]. Insightful (and more general) discussions of common sense, its implications, its nature and its manifestations can be found in [BRE91, FOR89, LUK90].

Before discussing reasoning, we first briefly address the problem of representation in the attribute-value language. Reasoning tasks are often expressed in natural language and have simple, direct counterparts in the classical first-order logic language (FOL). For PDL2 to reason about such first-order statements, they must first be translated into an equivalent symbolic-valued attribute language (AVL) representation. Note however that AVL is not as expressive as FOL, so that only certain FOL statements can be translated directly into AVL statements. Essentially, only implications whose predicates all range over a single universally quantified variable, and whose premise is a conjunction of predicates may be translated. In some cases, contextual information may be used to lift this restriction. Despite these limitations, several interesting commonsense reasoning protocols satisfy AVL's requirement, and can thus be handled by PDL2.

The mapping from FOL to AVL is as follows. Each "predicate" (e.g., on the table) becomes a Boolean attribute. Each "class" of objects (e.g., blocks) becomes a multi-valued (i.e., nominal) attribute. Each implication (or default rule) becomes a precept in which the attributes corresponding to the premise and the conclusion are asserted and all other attributes are don't-care. Following are two simple examples. The mapping of predicates to attributes ($attr_i$ and concl) is given below each example.

Example 1: Nixon Diamond

1. Republicans are normally not Pacifist	1. $1 * \rightarrow 0$
2. Quakers are normally Pacifist	2. $* 1 \rightarrow 1$

mapping: $attr_1$ = Republican, $attr_2$ = Quaker, concl = Pacifist

Example 2: Flying or Not Flying

1. Animals typically do not fly	1. $1 * * \rightarrow 0$
2. Birds are animals	2,3. $1\ 1 * \rightarrow 1$
3. Birds typically fly	
4. Penguins are birds	4,5. $1\ 1\ 1 \rightarrow 0$
5. Penguins do not fly	

mapping: $attr_1$ = Animal, $attr_2$ = Bird, $attr_3$ = Penguin, concl = Fly

Note that in some cases several implications are embedded in a single precept. This is an artifact of encoding that allows PDL2 to focus on the target (e.g., birds flying ability), and deal with the problem in its limited one-step inference context. However, it also tends to change the meaning of the corresponding English statements. For example, the statement "Animal birds typically fly," in PDL2's resulting representation, is different from the two statements "Birds are animals" and "Birds typically fly," in English. Extensions to the model, that support forward chaining and a more accurate representation, are under study.

3.1. Default Reasoning

PDL2's inherent mode of reasoning is default reasoning. Precepts given *a priori* and rules generated by induction from examples serve as defaults. They are always followed when matched by a new situation, unless the new situation also matches one of their exceptions. The default reasoning mechanism is supplemented by similarity-based reasoning, thereby decreasing the system's susceptibility to brittleness.

An important issue with default reasoning is that of conflicting defaults, as in the famous Nixon Diamond [REI81]. Such cases are characterized by the existence of two or more conflicting defaults that apply equally well to the situation at hand. In the absence of other information, the system is left in a "true" *don't know* state. However, it is our contention that such states should be minimized if a decision-making system (such as PDL2) is to be useful. In other words, if there exists some way to resolve the conflict, the system should make use of it. For two defaults to conflict, the sets of examples they represent must overlap. PDL2 offers two (possibly complementary) ways of effectively handling conflicting defaults. One is intensional, and the other is extensional.

The asserted inputs of a precept can be viewed as critical features characterizing a class of examples (namely those represented by the precept). Hence, precepts are a form of intension. Since PDL2 supports prior knowledge in the form of precepts, it can beneficially use the intensional approach. In this case, the system is given (*a priori*) the defaults and their relative priorities (based on some external source, e.g., religion is more important than politics). During an execution where several conflicting defaults apply equally well, all of the defaults have the same activation. If they also have the same generality (as in Example 1), PDL2 automatically causes the default with highest priority to win the competition. If they differ in generality, PDL2 chooses the more specific as the winner. Such an *a priori* ordering of defaults may not always be available, nor realistic however.

The other way to deal with conflicting defaults is extensional. PDL2's ability to learn inductively from examples allows the generation of default rules and/or the allocation of dynamic priorities to existing defaults based on extension (the set of training examples). Dynamic priorities are computed as follows. The priority value of a rule r is incremented every time r "sees" a pair p such that r covers p and p is an exception to at least one other rule. If p is an exception to another rule, say r', and r covers p, then r and r' conflict. We increment the priority of the rule that is matched by the incoming training pair. The priority value essentially gathers evidence in favor of one or the other conflicting default, and serves as a form of confidence level for that default. The system can then use this confidence level to make a decision. This decision is not certain; it is evidential.

Both approaches may be usefully combined by, for example, allowing observations to either confirm or refute "static" priorities. The system becomes inherently more adaptive.

3.2. The Brittleness Bottleneck

PDL2 handles many of the aspects of the brittleness problem, as characterized in [SUN92]. Its mode of execution, that combines rule-matching and similarity-based reasoning allows it to deal with a *lack of a matching rule*. When confronted with a new situation, the system is never left in a "don't know" state. A conclusion is always reached, either by applying a matched rule, or by using similarity with other learned situations.

If only *partial information* is provided, the missing pieces (attribute values in this case) are treated as "don't care" variables. In reasoning, the system makes its decision based on the information provided. If rules are available that encompass the partial information, then PDL2's conclusion is improved.

As shown above, PDL2's inductive learning ability allows it to naturally handle *generalization*. It is capable of learning new rules from examples, as well as modifying existing rules. PDL2 also always extends the partial information it is given to a total function from the input space to the output space. Every point of the input space is matched to some point of the output space, either via a rule that covers it or via its "proximity" to known points.

PDL2's learning is driven by the underlying assumption that rules (or precepts) are only typically true, and not necessarily always true. Hence, the search for and correct handling of exceptions are critical to the system's performance. Because of the way it deals with exceptions, PDL2 handles both linear and tree-structured *inheritance*, including exceptions to exceptions.

3.3. The Inheritance Problem

The inheritance problem is one of the most challenging problems of reasoning. It has received much attention and various solutions have been proposed [HOR90, SUN93, YOU93]. The combination of default (rule-based) reasoning and similarity-based reasoning in PDL2 provides a natural way (see [SUN92], Appendix A) of handling the problem of inheritance.

Inheritance implies the existence of some property of a class of objects and the ability of this property to be had (or inherited) by subclasses of objects (top-down), or superclasses of objects (bottom-up). Cancellation of inheritance occurs when exceptional subclasses exist that do not possess a property of the class they are derived from. When the inheritance hierarchy takes the form of a tree, it is called tree-structured inheritance.

In the context of this work, it is the output that characterizes the property. Classes of objects are represented by precepts or rules (see Example 2), and the counterpart of the subclass relation is the covers relation. Exceptions implement cancellation of inheritance.

In PDL2, exceptions to rules are always retained in the knowledge base. Rules or examples that confirm an existing rule (i.e., non-exceptional subclasses) are usually not retained in the knowledge base, unless they are found to be exceptions to other rules. It follows that the system also retains exceptions to exceptions.

This simple mechanism, combined with PDL2's execution giving priority to the more specific rules (i.e., subclass) accounts naturally for cancellation of inheritance and exceptions to exceptions. Top-down inheritance for non-exceptional subclasses is a direct result of the matching predicate (i.e., the node activation measure), since the precept representing the subclass matches exactly the precept representing the class it is derived from. Tree-structured inheritance follows from similar consideration applied to the branches of the tree. Bottom-up inheritance has not yet been investigated.

4. Simulation Results

A set of commonsense benchmark problems is proposed in [LIF88], and [MUR92] contains many useful training sets for inductive learning. This section reports results of simulations of PDL2 on several of these datasets, as well as preliminary results of the combination of inductive learning with prior knowledge and reasoning.

First, we show the inductive learning potential of PDL2, and the benefits gained by the addition of precepts on several datasets from [MUR92]. PDL2's measure of similarity (i.e., node activation) is designed to deal primarily with symbolic-valued (i.e., nominal) attributes. Consequently, the selected datasets have nominal attributes. All don't-know values are treated as don't-care values, except in the voting3-84 dataset in which they are treated as a distinct input value. The results are shown in Table 1. Each of the reported results is an average over 10 runs of PDL2. In each run, the training set and the test set are randomly regenerated. Each set contains one half of the examples in the complete data set.

The first number (PA) represents predictive accuracy (in %) on the test set after training. The second number (TS) is the number of instances used to train the system.

The third number (NS) is the size of the final network in nodes. The last column reports predictive accuracy and network size when precepts are added before training for five of the datasets. Currently, the precepts are obtained from domain knowledge provided with the application or generated from the authors' common sense. They are detailed in Appendix B. Other, more automatic ways of generating precepts are the topic of future research.

Table 1 - PDL2's Simulation Results

Application	PA	TS	NS	w/precepts
zoo	96.0	45	21	93.1 - 18
lenses	66.7	12	6	77.5 - 4
voting-84	79.4	217	86	82.0 - 50
voting3-84	92.2	217	148	94.7 - 60
tic-tac-toe	80.4	479	479	87.7 - 345
chess-ends	90.7	1598	932	No results
LED	65.8	500	125	Not applicable

Together with PA, the ratio NS/TS gives an idea of PDL2's generalization power. For the datasets above, it is .55 on average. The addition of precepts shows an increase of 5.65% in predictive accuracy, and a decrease of 35.4% of the network size on average. These results confirm the value of precepts as learning biases.

PDL2's reasoning scheme was tested against several of the benchmark problems of [LIF88]. As expected, PDL2 was able to successfully solve the following classes of problems.

- A1 - basic default reasoning
- A2 - default reasoning with irrelevant information
- A5 - default reasoning in an open domain
- A9 - priority between defaults
- B1 - linear inheritance (top-down)
- B2 - tree-structured inheritance
- B3 - one-step multiple inheritance

In addition, PDL2 solves the Nixon Diamond problem with either static priorities given *a priori*, or extensionally by using instances of Republican-Quakers to inductively learn the two contradicting defaults, and incrementally discover their relative priorities.

To further show the combination of inductive learning and reasoning, a simple, preliminary example of birds and penguins was developed. The application has four input attributes, two of which correspond to Penguin and Bird. The other two are other (undetermined) attributes of birds (e.g., Feather). The output is Boolean and characterizes the ability to fly. At first, the system is exposed mostly to penguin-birds. When asked whether birds typically fly (i.e., only the Bird attribute is asserted, and all other inputs are don't-care), PDL2 concludes that birds do not fly, which is consistent with its current experience with the "world." However, as more new instances of flying birds (i.e., other than penguins) are encountered, PDL2 adapts its knowledge and when asked again, concludes that birds fly. This simple case of bottom-up inheritance can thus be handled directly (B1), or extensionally.

5. Conclusion

We have described an incremental learning model, PDL2, whose ability to encode prior knowledge in the form of precepts is effectively supplemented by inductive mechanisms to learn from examples. The system is thus more self-adaptive in nature and matches more closely the way in which human learning takes place (as a combination of experience and theory). As a reasoning tool, PDL2 is shown to handle many challenging problems, including inheritance. It provides mechanisms to deal with many aspects of the brittleness problem, and is an attempt at combining inductive learning with prior knowledge and reasoning.

Current research includes:

- multiple outputs (to handle multiple defaults)
- forward chaining (e.g., by rebroadcasting)
- extension to more general spaces (e.g., continuous)
- better characterization of conflicting defaults (note that this is a difficult problem since "in the general case, it is not decidable whether two defaults are conflicting or not." [BRE91])
- representation language (attribute-value language limited) and generalization
- source, form, generation and use of prior knowledge

Appendix A

Following is the pseudo code for PDL2's learning algorithm. It is executed for each training pair *q*. The relations *equal* (eq), *subset* (sub) and *superset* (super) are the usual ones applied to the sets of examples represented by pairs. The relation *discriminated* (discr) means that the sets are disjoint. Two pairs that differ in exactly one asserted input value are *near match* (n_match), and *num_assert* is the number of asserted inputs in a pair. Finally, two pairs are *concordant* (conc) if they have the same output value.

```
Create a node new_node for q with e_flag = false
Set creat_flag = true and modif_flag = false
Phase 1: Broadcast q and gather new_node's e_flag
  Each node computes its matching state and e_flag
Phase 2: Broadcast new_node's e_flag and gather creat_flag, modif_flag
  For all nodes n (storing pair p) in the network
    Case super(n.p, q):
      If conc(n.p, q) then
        Increment cover_count(n)
        If e_flag(new_node) then increment pty(n)
        Else creat_flag = false
    Case eq(n.p, q):
      If conc(n.p, q) then creat_flag = false and update cnts(n)
```

Else   update cnts(n) and copy cnts(n) into cnts(*new_node*)
        If cnts(*new_node*) for q's output > 1 then creat_flag = false
  Case sub($n.p$, q):
    If conc($n.p$, q) then
      Increment cover_count(*new_node*)
      If e_flag(n) then increment pty(*new_node*)
      Else n self-deletes
  Case discr($n.p$, q):
    If conc($n.p$, q) & n_match($n.p$, q) & |num_assert($n.p$)-num_assert(q)| ≤ 1
      If size($n.p$) ≥ size(q) & num_assert($n.p$) > 1 then
        Create a copy of $n.p$ and drop the condition   (*)
        If not e_flag(n) then n self-deletes
        If not e_flag(*new_node*) then creat_flag = false
      Else if num_assert(q) > 1 then modif_flag = true
EndForAll   /* all nodes */
For all generalizations g created in (*)
  If not $\exists n$ s.t. conc($n.p$, g) & (eq($n.p$, g) or super($n.p$, g)) then
    Create a node for g
If modif_flag then
  Randomly select one of the nodes (*nn*) that set it
  Create a copy of q and drop the condition to get g
  If not e_flag(*nn*) then *nn* self-deletes
  If not $\exists n$ s.t. conc($n.p$, g) & (eq($n.p$, g) or super($n.p$, g)) then
    Create a node for g and retain *new_node*
Else if creat_flag then retain *new_node*
Else delete *new_node*

Due to the incremental nature of PDL2, and to limit the effects of ordering in learning, quantities are also updated for the training pair. This is reflected in the symmetry of the algorithm (e.g., subset and superset cases are reciprocal).

Appendix B

Following are the precepts used in the simulations reported in Section 4. We give them in terms of features and predicted output. All other attributes are set to don't-care.

<u>zoo dataset:</u>
1. If animal has four legs, then animal is a mammal
2. If animal has feathers, then animal is a bird
3. If animal lays eggs, is aquatic, and has fins, then animal is afish

<u>lenses dataset:</u>
1. If patient has low tear production rate, then patient is not fit for contact lenses

<u>voting and voting3 datasets:</u>
1. If representative voted 'no' on the 'physician-fee-freeze' issue, then representative is a democrat

tic-tac-toe dataset:

1. If top row of grid is 'XXX', then 'X' wins
2. If middle row of grid is 'XXX', then 'X' wins
3. If bottom row of grid is 'XXX', then 'X' wins
4. If top row of grid is 'OOO', then 'O' wins

The slightly lower predictive accuracy obtained on the zoo dataset with precepts may be due to the inherent uncertainty of the precepts used, or possible inaccuracies in the dataset itself.

CHAPTER 6
An Efficient Metric for Heterogeneous Inductive Learning Applications in the Attribute-Value Language*

An important characteristic of real-world applications is the mixture of nominal and linear (or ordered) attributes. Most learning algorithms (including PDL2) ignore this issue and use a homogeneous metric to compute similarity between two patterns. It does seem however that such homogeneous metrics are inappropriate for inherently heterogeneous spaces. This chapter investigates this issue further and proposes a metric that properly handles homogeneous spaces while being able to efficiently handle heterogeneous spaces.

The most interesting aspect of the proposed metric is its ability to handle don't-cares, as well as continuous attributes directly (i.e., no need for discretization). This work was developed in parallel with the design of ILA (see Chapter 7 and [GIR94d]), and a simplified version of ILA was used to empirically evaluate the new metric. Section 6 is not in the published paper, but is relevant here as it discusses the heterogeneous metric in the light of the combination of learning and reasoning. Some of the details and examples presented here are not in the paper either due to a strict page limit. The format has also been significantly altered.

The metrics discussed here are independent of the context, that is they are insensitive to the set of surrounding points. In other words, if X and Y are two sets of points such that $X \neq Y$ and if p and q are two points such that $p,q \in X$ and $p,q \in Y$, then $d(p,q)$ in X is equal to $d(p,q)$ in Y. One could argue that better results may be obtained with context-sensitive metrics. One such metric is proposed in [CHE91].

1. Introduction

Many inductive learning systems use the classical attribute-value language as their representation language. In the attribute-value language, a training example is a vector whose entries are pairs consisting of an attribute name and its value. In most cases, attribute names are omitted as the context unambiguously determines which entry corresponds to which attribute. Some of the attributes are identified as being input attributes, and some (often a single one) are identified as output attributes. Each input attribute's value ranges over some domain, and the cross product of all the domains defines the input space of the application. Hence, training examples are points in the input space, labeled with some output value.

In many cases, inductive learners are concerned with the classification of points into disjoint categories. The system uses training examples to derive a classification that is then generalized to previously unseen points of the input space. Generalization typically consists of either one (or a combination) of two processes. On one hand, the system

---

 To appear in *Proceedings of the Third Golden West International Conference on Intelligent Systems (GWIC'94)*, Las Vegas, NV, June 1994, 1-10 (??). Permission pending.

attempts to discover subsets of points (or regions) of the input space that map to a given concept [DIE83, GIR93b]. On the other hand, the system uses proximity (or similarity) to known points [AHA91b, GIR94d, KIB87, SUN92]. When a new point is shown, its classification is determined either by its belonging to one of the identified regions, or by the classification of the "closest" known points. Classification by closeness is the essence of most nearest-neighbor algorithms (see [DAS91] for a survey) and memory-based reasoning (see for example [STA86]).

To find the "closest" known points to a new point in the input space, the system needs some measure of proximity or similarity, i.e., a distance metric defined on the input space. For example, when dealing with applications whose attributes are all Boolean, the classical Hamming distance could be used as a measure of closeness. The larger the distance between two examples, the fewer features the two examples have in common. The problem, of course, is that the Hamming distance is unlikely to give useful results for an application whose attributes are all continuous. In this case, the Euclidean distance could be used. Yet, the Euclidean distance is unlikely to produce useful results for an application whose attributes are all nominal. Such homogeneous distance measures seem to be useful only as long as all attributes are of the same type.

Because the domains of attributes vary in nature, the input space is often heterogeneous, in the sense that topological properties such as distance do not have the same definition in all dimensions. The notion of distance between two points in such heterogeneous spaces is non-trivial. This chapter discusses the limitations of homogeneous metrics, and proposes a (combined) heterogeneous distance measure that accounts for heterogeneous spaces, in the context of inductive learning. Empirical results demonstrate the superiority of the proposed metric over homogeneous distance measures.

Section 2 addresses the issue of similarity, discusses two representative homogeneous metrics, and gives examples of the inadequacy of these metrics in heterogeneous domains. Section 3 discusses two issues raised by the design of a metric for heterogeneous learning applications. Section 4 proposes a heterogeneous metric that extends the similarity function of [AHA91b]. Section 5 overviews a simplified version of ILA [GIR94d] that serves as a basis for empirical evaluation. Section 6 briefly addresses the impact of the new metric on attempts at combining learning and reasoning. Finally, Section 7 concludes the chapter.

2. Considerations on the Notion of Similarity

We consider here two types of attributes: nominal and linear. Nominal attributes have discrete values that are not related in any way other than the fact that they belong to the same set. Linear attributes have values that can be ordered in the usual sense, and whose ordering is relevant to the context in which they are used. Linear attributes may be discrete and ordered or continuous. The attribute *blood group*, for example, is a nominal attribute, while the attribute *weight* is linear. Each kind of attribute gives rise to a distinct notion of similarity or distance. We discuss two of them here, and show that they are mutually incompatible. The selected metrics are in no wise unique, only representative. They serve as illustration, are commonly used [AHA91b, GIR93b, SUN92], and have been found to give good empirical results on their respective domains of applications (see Section 5).

2.1. Nominal Attributes and Distance

For nominal data, the notion of how far apart two values are reduces to a simple binary relation. Two values are either the same, or they are different. If the values are the same, then the distance is 0, and if they are different, the distance is 1. This one-dimensional definition is extended to the multi-dimensional case in a straight forward way. The distance between two vectors of nominal attributes is the sum of the distances of their attributes. Formally,

$$\text{Let } \boldsymbol{x} = (x_i) \text{ and } \boldsymbol{y} = (y_i) \text{ be two } n\text{-dimensional nominal vectors}$$

$$\text{Let } dn(x_i, y_i) = \begin{cases} 1 & \text{if } x_i \neq y_i \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Then } DN(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} dn(x_i, y_i)$$

DN (Distance for Nominal spaces) conveys the intuitive idea that the farther away two vectors are, the less similar they are. So it can be used directly to choose a closest match.

2.2. Linear Attributes and Distance

Linear attributes are ordered (e.g., continuous values). The ordering gives rise to a natural measure of distance, namely, the farther away things are in the ordering, the larger their distance should be. This is consistent with our everyday notion of distance. For example, the (natural) numbers 2 and 5 are further apart than the numbers 2 and 3. In this case, the distance between two values can be defined as the absolute value of their difference. Since every ordered set is in one-to-one correspondence with a subset of the natural numbers, this distance is well-defined on all domains (non-numeric values are translated into their equivalent numeric form via the correspondence). A common generalization of this one-dimensional definition to the multi-dimensional case is the classical Euclidean distance.

A problem with this definition is that it implicitly assumes that all attributes range over the same domain (e.g., reals, integers). If the domains are different, then some attribute distances may dominate others in the overall distance. For example, if x and y are linear attributes ranging over [0, 1] in the real numbers and [0, 100] in the natural numbers, respectively, then distances along y are likely to dominate distances along x. The smallest distance in y is equal to the largest distance in x. The problem is one of scale. We thus modify the definition as follows.

First, let us argue that in most practical learning applications, linear attributes are bounded, that is they have a smallest and a largest possible value. These can easily be obtained from the associated dataset. To eliminate the effects of statistical outliers, the dataset must be ridden of examples whose attributes have such "irregular" values. Alternatively, the linear attributes (especially continuous ones) can be discretized into finitely many classes.

Then, let *range(i)* denote the range of values for attribute i, that is, the difference between the maximum and minimum values of attribute i. We now define *NDL* (Normalized Distance for Linear spaces) by:

$$\text{Let } \boldsymbol{x} = (x_i) \text{ and } \boldsymbol{y} = (y_i) \text{ be two } n\text{-dimensional linear vectors}$$

$$\text{Let } nd(x_i, y_i) = \frac{|x_i - y_i|}{range(i)}$$

$$\text{Then } NDL(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{n} [nd(x_i, y_i)]^2}$$

If *range(i)*=0, i.e., attribute i has the same value in all vectors, then attribute i is ignored in the computation of *NDL* (i.e., $ndl(x_i,y_i)=0$ for all $\boldsymbol{x}$ and $\boldsymbol{y}$). The division by *range(i)* causes all attribute distances to fall within the range [0, 1]. Hence, all attributes make a normalized contribution to *NDL*. Again, *NDL* conveys the intuitive idea that the farther away two vectors are, the less similar they are. So it can be used directly to choose a closest match.

2.3. Inadequacy of Homogeneous Metrics on Heterogeneous Spaces

In this section, we give simple examples of the inadequacy of *DN* on linear spaces and the inadequacy of *NDL* on nominal spaces, thus demonstrating the need for heterogeneous metrics on heterogeneous spaces. We also discuss some of the issues raised in designing such a heterogeneous metric.

2.3.1. Inadequacy of DN on a Linear Space. Consider the following example, where each attribute ranges over [0, 10] in the natural numbers.

$$\begin{aligned} x &= (2,1,3) \\ y &= (3,1,4) \Rightarrow DN(x,y) = 2 \\ z &= (2,6,0) \Rightarrow DN(x,z) = 2 \end{aligned}$$

Though $DN(x,y)=DN(x,z)$, it appears that (given linear attributes) y is closer to x than z is. The problem is that the ordering implies varied magnitudes in the differences between values, while *DN* only accounts for equality or inequality. It takes any magnitude greater than 0 to 1, and leaves 0 magnitudes at 0. In some sense, it forces a step function on a linear domain. Things become even worse in continuous spaces where the probability of any two values being equal is extremely low.

2.3.2. Inadequacy of NDL on a Nominal Space. Consider the following example, where each attribute ranges over the discrete set {0, 1, 2, 3, 4, 5}. For a nominal domain, we let *range(i)* be the number of possible values in the domain minus 1.

$$x = (3,1,2)$$
$$y = (2,1,1) \Rightarrow NDL(x,y) = \sqrt{2}/5$$
$$z = (3,4,2) \Rightarrow NDL(x,z) = 3/5$$

Though $NDL(x,y)<NDL(x,z)$, it appears that (given nominal attributes) z is closer to x than y is. The problem is that *NDL* treats nominal values as if they had inherent magnitudes, whereas they do not. Instead of concluding that the distance between any two different values is the same, it assigns different distances based on some non-existent, but assumed, intrinsic ordering.

One may be tempted to claim that the above example is a result of an artifact of encoding. If all nominal values are "one-hot" encoded (i.e., as many bits as there are values, one set for each distinct value), then *NDL* would make z closer to x as expected. However, such encodings cause a multiplication of the number of attributes. Moreover, they reduce nominal spaces to Boolean spaces, and it is straight forward to show that on Boolean spaces $NDL=DN$.

2.3.3. Inadequacy of NDL and DN on a Heterogeneous Space. The above examples are based on the application of a homogeneous metric to a homogeneous space. They show that if a learning system uses only one metric and applies it in a non-discriminatory way to all applications, then it is likely that its accuracy will suffer on some applications. It is clear that similar results hold when applying either of the homogeneous metrics to a heterogeneous space. Consider the following example, where the first attribute is nominal and ranges over the discrete set {0, 1, 2, 3, 4, 5}, and the other two attributes are linear and range over [0, 10] in the natural numbers.

$$x = (4,1,0)$$
$$y = (4,7,0) \Rightarrow NDL(x,y) = 6/10 \wedge DN(x,y) = 1$$
$$z = (3,0,2) \Rightarrow NDL(x,z) = \sqrt{6}/10 \wedge DN(x,z) = 3$$

The two distances naturally give inconsistent results. *DN* leads to the conclusion that y is closer to x than z is, while *NDL* leads to the opposite conclusion. This inconsistency is a direct consequence of the problems discussed in Sections 2.3.1 and 2.3.2. It follows that to correctly handle heterogeneous spaces, a heterogeneous metric must be employed.

3. Designing a Metric for Heterogeneous Learning Applications

We address two important issues raised by the design of a metric for heterogeneous learning applications. One has to do with the heterogeneity of the learning space, and the other is a result of the use of rules containing *don't-care* values in many learning algorithms.

3.1. Heterogeneity of the Learning Space

In designing a metric for heterogeneous spaces, it is desirable not to loose accuracy on homogeneous spaces. A simple solution consists of combining the homogeneous metrics into a single metric that applies the nominal metric to nominal attributes and the linear metric to linear attributes. A potential problem with such a combination is the scale problem.

In this case, the problem is no longer with each attribute's range, but with the ranges of values covered by the homogeneous metrics. If these are different, then one of the metrics may indeed dominate the overall result. We illustrate this problem by considering a metric *DLDN*, that applies *dn* on nominal attributes and a non-normalized version of *ndl* on linear attributes, as follows.

$$DLDN(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} \begin{cases} dn(x_i, y_i) & \text{if attribute } i \text{ is nominal} \\ |x_i - y_i|^2 & \text{if attribute } i \text{ is linear} \end{cases}}$$

Consider the following example, where the first two attributes are Boolean, and the last two are linear ranging over [0, 10].

$$\begin{aligned} &x = (0,1,3,2) \\ &y = (1,1,3,2) \Rightarrow DLDN(x,y) = 1 \\ &z = (0,1,4,2) \Rightarrow DLDN(x,z) = 1 \end{aligned}$$

Though the distance between x and y is the same as the distance between x and z, it can be argued that z is closer to x than y is. Indeed, in the first case, y and z differ on the value of a Boolean attribute, thus a complete mismatch. In the second case, x and z differ in the value of a linear attribute, and the difference is small (given the range of possible values). One says that y has a given property, while x does not, and the other says that z has a little bit more of a property than x has. Having less of something does not seem nearly as bad as not having it at all. The problem is that *DLDN* has an upper bound of 1, on the distance between nominal values, but a potentially much larger bound on the distance between linear values. The simple combination does not appear satisfactory, even though each metric separately is a valid candidate for the corresponding homogeneous spaces.

Hence, it is not sufficient that each metric be adequate on its own space. They must somehow be normalized with respect to each other. The normalization in *ndl* causes *ndl* to have exactly the same range of values as *dn*, namely [0, 1]. Hence, not only is *ndl* a more accurate linear metric (for linear spaces), it is also "compatible" with *dn*.

There is a possible problem with such a normalization however. Indeed, by normalizing, we artificially force complete mismatches (i.e., distance of 1) to the extrema. For example, if an attribute ranges from 0 to 1000, only values close to 0 compared with values close to 1000 produce distances close to 1. If we are to compare the values 0 and 500, then after normalization we are left to consider the values 0 and .5. In the latter case, the values do not seem to be as far apart as they are in the former. It would be reasonable to assume, however, that when two values are far enough apart (based on some threshold value), then there is a complete mismatch between them. Hence, instead of pushing mismatches to the extrema, it is possible to choose threshold values for the differences between linear values, beyond which the distance is set to 1 (i.e., mismatch). In other word, a non-linear normalization technique may be more appropriate. Another alternative consists of using multiplicative scale factors on nominal attribute distances. We do not address this problem in this chapter.

3.2. Using Rules with Don't-Care Values

A common and useful extension, made to the classical attribute-value language, is the addition of a special *don't-care* symbol for input attributes. The interpretation for a vector containing input attributes set to don't-care is as follows. An attribute whose value is not don't-care is said to be *asserted*. A vector whose input attributes are all asserted represents a single point of the input space. A vector that has input attributes set to don't-care represents all of the points in the input space that would be obtained by replacing all of the don't-care symbols by any one of the possible values of the corresponding input attributes. Consider the following example where each attribute ranges over the discrete set {1, 2, 3, 4}.

$$\begin{array}{l} \textit{Let } \boldsymbol{v} = (1,*,3,2) \\ \textit{Then } \boldsymbol{v} \textit{ represents the following points} \\ (1,1,3,2),\ (1,2,3,2),\ (1,3,3,2),\ \textit{and } (1,4,3,2) \end{array}$$

Hence, such vectors represent subsets of points, or regions, in the input space. We discuss two situations in which they arise, and show the impact they have on the definitions of distance.

3.2.1. Origins of Don't-Cares. Vectors with input attributes set to don't-care represent subsets of points in the input space. Hence, they can be viewed as *rules* in the sense that they map a set of points, rather than a single point, to a given output value (or concept). The number of points represented by a rule is a measure of that rule's *generality*. The larger the set of points, the greater the generality. Such rules arise in at least two situations that are both relevant to learning systems.

The purpose of inductive learning systems is to extract, from a set of training examples, the critical features that govern an application. Some of these critical features, or set of attribute-value pairs that are good predictors of the output, may be represented as rules. Such rules are the result of the learning system's ability to generalize. A commonly used generalization rule in learning systems is the *dropping-the-condition* rule [MIC83a]. In the attribute-value language, it translates into the setting of one (or more) of the input attributes of a given vector to don't-care. The input attributes that remain asserted can then be viewed as critical features (those that are don't-care do not affect the output value).

Another (possibly complementary) situation in which rules arise is by being provided *a priori* (by some teacher). Indeed, such rules may also be used as an encoding of prior knowledge, or as a particular instantiation of domain knowledge or common sense. They then serve as learning biases for the system (see, for example [GIR93b]). For example, when learning about the flying abilities of birds, one may bias the system with a rule encoding the fact that birds typically fly. The system then only needs to focus on exceptions.

3.2.2. Effects of Don't-Cares. Don't-cares (regardless of their origin) have a direct impact on the definitions of distances on the input space. Since they do not have any intrinsic value, they are not comparable with other values. To find a closest vector in the current knowledge base, distance is measured from vectors in the knowledge base to new vectors. If $\boldsymbol{x}$ and $\boldsymbol{y}$ are two vectors such that all of their attribute values are equal except for some attributes that are don't-care in $\boldsymbol{x}$ and asserted in $\boldsymbol{y}$, and if d is any well-defined distance, then we claim that the following facts should hold.

$$d(\boldsymbol{x}, \boldsymbol{y}) = 0 \qquad (1)$$
$$d(\boldsymbol{y}, \boldsymbol{x}) \neq 0 \qquad (2)$$

That is, the distance measure is not symmetric. In (1), $\boldsymbol{x}$ is in the current knowledge base, $\boldsymbol{y}$ is a new vector, and the equality is an immediate (and natural) consequence of the definition of don't-cares. In (2), $\boldsymbol{y}$ is in the current knowledge base, $\boldsymbol{x}$ is a new vector, and the inequality requires some explanation.

If reasonable at all, inequality (2) is only useful when unseen vectors (either in training or in testing) are allowed to have don't-care input attributes. We argue that such situations may arise and claim that in such cases, the inequality should hold. As seen in Section 3.2.1., rules may be the expression of domain knowledge. There is no reason to assume that all of these are available *a priori*. It is plausible that the learning system (much like a human in this case) is exposed to examples and rules in an unpredetermined order. Hence, certain rules may come in later in the learning process. For example, if you live in Australia, most of the birds you see may be ostriches that do not fly. Thus, you are exposed to exceptions before you may be taught the more general rule about birds flying ability. In testing, allowing don't-cares is a way to potentially discover default rules for situations in which much information is missing. This is particularly useful in bottom-up inheritance, where, based on the current knowledge base, one can obtain the default classification of a very large set of points. In this latter case, a don't-care can also be thought of as a don't-know or missing value.

If (1) holds and the distance is symmetric, then the generality of the new vector may cause several vectors in the current knowledge base to be equidistant to it, as in the following example, where all attributes are Boolean, and the last one is the output. The knowledge base is KB, the new vector is $\boldsymbol{x}$, and * stands for don't-care.

$$KB = \{(1,1,0,1,1), (1,1,*,1,0), (*,0,*,1,1)\}$$
$$\boldsymbol{x} = (1,*,*,0,?)$$

All of the vectors in KB have the same distance to $\boldsymbol{x}$. However, in this case, where a default classification is expected, it seems reasonable to expect the more general of the vectors in the current knowledge base to be closest. Inequality (2) increases the distance to more specific existing vectors, thus causing the system to favor the more general vectors.

Another impact of the variation in generality of different vectors and (1) is the need for some normalization of the overall distance, so as to distinguish a perfect match from a don't-care value. Because attributewise distances range over [0, 1], this can simply be accomplished by dividing the overall distance by the number of asserted input attributes in the vector from which distance is computed. For example, it is better to have 2 mismatches out of 5 asserted input attributes than 1 mismatch out of only 2 asserted input attributes.

4. DHET - A Metric For Heterogeneous Inductive Learning

Based on the above discussion, we propose the following modified set of definitions. Let *num_asserted(**x**)* be the number of asserted input attributes in vector ***x***.

$$\text{Let } \mathbf{x} = (x_i) \text{ and } \mathbf{y} = (y_i) \text{ be two } n\text{-dimensional vectors}$$

$$\text{Let } dn'(x_i, y_i) = \begin{cases} 1 & \text{if } x_i \neq * \wedge x_i \neq y_i \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Then } DN'(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^{n} dn'(x_i, y_i)}{num\_asserted(\mathbf{x})}$$

$$\text{Let } nd'(x_i, y_i) = \begin{cases} \dfrac{|x_i - y_i|}{range(i)} & \text{if } x_i \neq * \wedge y_i \neq * \\ 1 & \text{if } x_i \neq * \wedge y_i = * \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Then } NDL'(\mathbf{x}, \mathbf{y}) = \frac{\sqrt{\sum_{i=1}^{n} \left[nd'(x_i, y_i) \right]^2}}{num\_asserted(\mathbf{x})}$$

$$\text{And finally, } DHET(\mathbf{x}, \mathbf{y}) = \frac{\sqrt{\sum_{i=1}^{n} \begin{cases} dn'(x_i, y_i) & \text{if attribute } i \text{ is nominal} \\ nd'(x_i, y_i)^2 & \text{if attribute } i \text{ is linear} \end{cases}}}{num\_asserted(\mathbf{x})}$$

The value of 1 for *dn'* and *ndl'* when x_i is asserted and y_i is not reflects the fact that we consider such a case to be a "complete" mismatch. *DN'* is the reciprocal of the conceptual similarity measure of [SUN92]. *DHET* (Distance for HETerogeneous spaces) is an extension of IBL's similarity function [AHA91b]. The two metrics are equivalent when a purely instance-based learning algorithms is used, and *DHET* extends IBL's similarity function to inductive learning algorithms that use and/or create general rules. *DHET* is also similar to the distance metric used by NGE [SAL91]. It is somewhat simpler since rules containing don't-cares represent hyperplanes rather than hyperrectangles. It is also inherently heterogeneous whereas NGE's distance is designed for purely continuous domains. Extensions to heterogeneous spaces have been proposed (see for example [WET93]). NGE's distance is also a weighted sum, where each attributewise distance is assigned its own weight in the computation of the final distance. Mechanisms to assign weights to attributes in distance computation may be found in [SAL91, STA86, WET93]. *DHET* could be similarly extended.

5. Evaluation

In this section, we give empirical evidence of the adequacy of *DHET*. The results were obtained by executing the same algorithm, only varying the metric it uses. ILA (Incremental Learning Algorithm) was chosen because of its simplicity, execution speed, and the direct impact the metric has over its accuracy. ILA bears resemblance with IBL algorithms [AHA91b] and NGE [SAL91]. It can be viewed as a nearest-hyperplane algorithm. We first overview a simplified version of ILA. Details of the full algorithm are in [GIR94d]. We then present results of simulations on a variety of applications.

5.1. ILA - Algorithmic Overview

ILA's representation language is the classical attribute-value language, with the addition of the special *don't-care* symbol. For convenience, examples and rules are viewed as *input-output pairs*. Examples have all of their input attributes asserted to some value. ILA discovers new rules by generalizing from examples. The generalization rule it uses is the classical drop the condition rule [MIC83a], which causes the value of an input attribute to be set to don't-care.

ILA implements a network of simple nodes that adapts to newly acquired knowledge by dynamically adding and/or removing nodes in the network. The number of parameters to be set *a priori* is kept to a minimum, and the final network is the result of a sequence of self-organizing transformations. The network's structure is a balanced binary tree. ILA's learning algorithm requires only a simple gather and broadcast mechanism. Parallelism at the node level is inherent.

For our purposes, each node stores a pair, together with counters for each output value (vector *counters*), and a counter for the number of pairs covered (*num_covers*). The counters are used to handle noise. For each output value, they keep track of the number of training pairs whose inputs match exactly the input of the node's stored pair. The output whose count is highest is the node's output. The counter num_covers keeps track of the number of pairs that have the same output as the stored pair, and are covered by it. We say that a pair *p covers* a pair *q* if and only if the set of points represented by *q* is a proper subset of the set of points represented by *p*. The value of num_covers can serve as the degree of confidence in the node's stored rule.

ILA's learning consists of two phases: *execution* and *adaptation*. In the execution phase, the training pair is presented to the network and a winning node is identified. In the adaptation phase, counters and num_covers are updated, and a new node may be created for the training pair. The decision to create a new node is based on the relationship of the winning node's stored pair to the training pair. The algorithms given below are repeated for each new training pair *Q*.

Execution Phase
Broadcast(*Q*)
Identify a winning node *W* (storing pair *P*)

Identifying the winner is a simple gather operation, that first minimizes the distance (i.e., node with smallest distance wins), then resolves ties by giving priority to the more specific node (i.e., larger number of asserted inputs), then to the node with the largest value of num_covers. Winning distances are passed up and nodes are tagged as self, right or left, based on which node wins the competition. A winner is then easily identified by going down the tree until a node marked as self is found. If the network is empty, then *W* does not exist, and a new node is automatically added for the training pair. Giving priority to more specific nodes helps in handling exceptions, and giving priority to nodes whose num_covers value is higher increases the chances of selecting a "most likely to be correct" winner.

Learning Phase
```
diff = | num_asserted(P) - num_asserted(Q) |
If (equal(P, Q)) then
  increment P.counters(Q.output)
else if (superset(P, Q) and same_out(P, Q)) then
  increment P.num_covers
else if (discriminated(P, Q) and same_out(P, Q) and near_match(P, Q) and diff ≤ 1) then
  if (P.size ≥ Q.size and P.num_asserted > 1) then
    generalize P
  else if (Q.num_asserted > 1) then
    generalize Q
    create a node for Q
else if (subset(P, Q) and same_out(P, Q)) then
  increment Q.num_covers
  delete W
  create a node for Q
else
  create a node for Q
```

As nodes are added or removed from the network, the binary tree structure is kept balanced. After all training pairs have been seen, ILA checks every node, and sets its output to the value whose counter is highest. This simple mechanism enables ILA to deal with noise. The inherent symmetry in the "equal" case, and between the "superset" and "subset" cases, is a reflection of ILA's attempt at reducing the impact of ordering on learning.

Evaluating ILA for predictive accuracy on a test set consists of presenting every pair in the test set and only running the execution phase. The output of the winning node is the system's output and prediction of the new pair's output.

5.2. Simulation Results

A variety of datasets from the Irvine repository of machine learning datasets [MUR92] were chosen, representing a mixture of homogeneous and heterogeneous applications. Two modifications are made however. All continuous attributes are made discrete (using a K-means algorithm on each input), and all don't-know values are arbitrarily treated as don't-care values. Other (possibly more appropriate) ways of handling don't-know values are the topic of future research (three of them are discussed in [AHA91a]). Also, even though the attributes of the dataset lenses are all claimed to be nominal, the first one (i.e., age) appears to have a definite natural ordering to it. Hence, the dataset lenses is considered to be heterogeneous. Tables 1 through 3 list the attributes of the homogeneous nominal datasets, the homogeneous linear datasets, and the heterogeneous datasets, respectively.

Table 1 - Homogeneous Nominal Applications

Dataset	#instances	#inputs	#out. values
zoo	90	16	7
mushroom	8124	22	2
chess-end	3196	36	2
molecular-bio	106	57	2
tic-tac-toe	958	9	2
LED	1000	7	10

Table 2 - Homogeneous Linear Applications

Dataset	#instances	#inputs	#out. values
iris	150	4	3
glass	214	9	7
breast-cancer	699	9	2
pima-indians	768	8	2
vowel	528	10	11
sonar	208	60	2

Table 3 - Heterogeneous Applications

Dataset	#instances	#inputs(N/L)	#out. values
lenses	24	3/1	3
hepatitis	155	13/6	2
soybean-sm.	47	31/4	4
credit-screen.	690	8/6	2
australian-crx	690	8/6	2

ILA's predictive accuracy (in %) on a test set was gathered for each of the above applications. Each of the reported results is an average over 10 runs of ILA for that application. In each run, the training set and the test set are randomly regenerated. Each set contains one half of the examples in the complete dataset. Results of simulations with *DN'*, *NDL'*, and *DHET* are given in Tables 4 through 6 for each kind of application.

Table 4 - ILA - Nominal Applications

Application	DN'	NDL'	DHET
zoo	95.3	94.7	95.3
mushroom	100	100	100
chess-end	90.5	90.1	90.5
molecular-bio	77.9	66.8	77.9
tic-tac-toe	80.4	75.8	80.4
LED	70.2	70.2	70.2
Averages	85.7	82.9	85.7

As expected, the results with *DHET* are identical to the results with *DN'*. The difference between *NDL'* and *DN'* on nominal applications is expected to be the greatest for applications whose attributes all have more than two possible values (e.g., the molecular-biology dataset). Indeed, it is straightforward to show that, on Boolean domains, *DHET=NDL'=DN'*.

Table 5 - ILA - Linear Applications

Application	DN'	NDL'	DHET
iris	90.5	94.0	94.0
glass	66.4	67.9	67.9
breast-cancer	94.4	94.7	94.7
pima-indians	63.4	67.2	67.2
vowel	78.2	91.9	91.9
sonar	79.8	85.4	85.4
Averages	78.8	83.5	83.5

As expected, the results of *DHET* and *NDL'* are identical. Also, the difference between the performance of *NDL'* and *DN'* is significant (almost 5% on average).

Table 6 - ILA - Heterogeneous Applications

Application	DN'	NDL'	DHET
lenses	65.0	80.0	80.0
hepatitis	77.3	78.6	78.6
soybean-sm.	98.8	98.8	98.8
credit-screen.	79.5	80.3	80.5
australian-crx	79.9	82.2	82.0
Averages	80.1	83.9	84.0

It appears that, on average, *DHET* is similar to *NDL'*. This may be explained by the fact that most heterogeneous applications have nominal attributes that are actually Boolean, or that have a small number of possible values. The results still show that *DHET* tends to give the best of both worlds on heterogeneous applications. One notable exception was encountered with the heart dataset (270 instances, 13 inputs: 7L/8N, 2 output values), where *DN'* gives 78.2%, *NDL'* gives 75.0%, and *DHET* only gives 75.7%. The reasons for such a behavior are unclear. Table 7 summarizes the averages on each kind of application and shows the overall averages for all applications.

Table 7 - ILA - Summary Results

Space	DN'	NDL'	DHET
Nominal	85.7	82.9	85.7
Linear	78.8	83.5	83.5
Heterogeneous	80.1	83.9	84.0
Overall	81.6	83.4	84.4

It is clear that each homogeneous metric outperforms the other on the corresponding spaces. Also, the results suggest that *DHET* is an adequate metric for learning applications in the attribute-value language. Over the selected datasets, representing a variety of input spaces, the average predictive accuracy obtained with *DHET* exceeds the average predictive accuracy obtained with either of the homogeneous metrics discussed here.

6. Combining Learning and Reasoning

One of the motivations for designing ILA was to attempt to combine learning and reasoning into a unified framework. Such a combination can prove useful when dealing with applications for which some domain knowledge is available, or in commonsensical situations. The idea is that some of the domain knowledge or common sense may be encoded in the form of rules, that can then be used to bias the system (e.g., prune the search for generalizations in the input space). A more thorough discussion on the topic is found in [GIR93b].

For the purposes of the present discussion, the critical issue is that many commonsense protocols, whose representations are often symbolic in nature (e.g., first-order language), are easily translatable into homogeneous nominal applications in the attribute-value language. Essentially, each predicate gives rise to a Boolean attribute, each class of objects gives rise to a nominal (multi-valued) attribute, and each implication (or default) gives rise to a rule (i.e., a pair with some input attributes set to don't-care).

Moreover, similarity measures such as *DN'* have proven useful in accounting for many important commonsense protocols, such as top-down inheritance, bottom-up inheritance, and cancellation of inheritance (see for example [GIR94b, SUN93]). In Section 4, we discussed some of these issues and showed how the metrics had to be modified to properly handle such cases. The metric *DHET* that we propose is equivalent to *DN'* on homogeneous nominal spaces and produces better results on average on homogeneous linear and heterogeneous spaces. Hence, it is a viable metric for systems that attempt to combine inductive learning in general domains and reasoning. Examples of such systems include ILA [GIR94d] and FLARE [GIR94e].

One further advantage of *DHET* that results from its non-symmetric definition is discussed in Section 3.3.2 of Chapter 8, where it is shown that the useful combination of rule-based reasoning and similarity-based reasoning can be achieved simply by minimizing *DHET*.

7. Conclusion

In this chapter, we have addressed the problem of designing an efficient metric for heterogeneous inductive learning applications in the attribute-value language. We gave examples of the inadequacy of homogeneous metrics, and proposed a metric, *DHET*, that effectively handles homogeneous and heterogeneous spaces. The need for normalization was discussed, and *DHET* was extended to account for learning algorithms that deal with rules containing don't-cares. Results of simulations with a simplified version of the algorithm ILA were given, that confirm these findings. On average, the results with the heterogeneous metric exceed the results obtained by homogeneous metrics on all spaces. Because *DHET* is identical to a classical homogeneous metric on nominal spaces, it can also be effectively used in systems that attempt to combine learning and reasoning.

The metric *DHET* is not claimed to be optimal in any sense. It does appear however to be a viable candidate for broadening the range of applications efficiently handled by current learning systems. Because *DHET* is not bound to any learning paradigm, it can potentially serve as an alternative metric in other existing systems. The only assumption necessary to *DHET* is the *a priori* knowledge of the types of the attributes used. Such knowledge is likely to be available at no cost, and it is assumed (sometimes only implicitly) in most learning algorithms.

Future research includes the following extensions to *DHET*:

- Normalization based on standard deviations, rather than ranges of possible values
- Non-linear normalization for linear attributes
- Individual weighting of features (likely to be dependent upon the learning algorithm)
- Adequate treatment of *don't-know* values. Treating don't-know (i.e., missing) values as don't-cares is a first attempt, however it is likely to produce unwanted results. One possibility consists of treating don't-know values as distinct attribute values (see also [AHA91a]).

CHAPTER 7
ILA: Combining Inductive Learning with Prior Knowledge and Reasoning[*]

In this chapter, we introduce another model, called ILA (Incremental Learning Algorithm), that attempts to combine inductive learning with prior knowledge and reasoning. The new model is compared to PDL2. Its advantages include the use of a heterogeneous metric (extension of DHET to missing attributes, Chapter 6), more potential for incrementality, greater predictive accuracy, and a somewhat simpler learning algorithm. The reasoning power remains the same as that of PDL2 however. A better characterization of ILA's reasoning is given, and several of its critical properties (see Chapter 4) are discussed in detail.

1. Introduction

Machine Learning has recently emerged as a promising alternative to programming for the implementation of solutions to many real-world problems on computers. The learning approach offers many advantages over the programming approach, particularly in contexts where information is uncertain or incomplete. Indeed, programming presupposes that all of the knowledge necessary to the solution of a given problem be available a priori. This assumption is valid in many well-defined, closed domains, such as mathematics or accounting. However, there are many situations in which the very nature of the knowledge (at least the representation thereof) is incomplete and uncertain, such as medical diagnosis, pattern recognition, geological prospection, and many decision-making applications (e.g., loan underwriting, mine-rock discrimination).

Whereas solutions to problems in these domains are difficult to program, learning may offer a viable and effective alternative. In most inductive learning systems, human interaction is typically limited to the gathering and compilation of examples (i.e., instances of the application to be learned). As research on autonomous agents continues, this process itself may eventually become automated. However, the strong knowledge principle [WAT86], and early work on bias [MIT80] suggest that examples should be augmented by prior knowledge. Human learning, for example, is not the sole result of exposure to random examples. Rather, built-in mechanisms (e.g., pain), and social structures (e.g., the family, school) account for much of humans ability to efficiently learn complex problems. In many cases, useful information is indeed available as, for example, an instantiation of domain knowledge or common sense. Though such knowledge is not necessarily correct, it can beneficially be used as a learning bias to supplement inductive mechanisms (see, for example [GIR93b]). As a first attempt (and currently the only viable one), this prior knowledge can be obtained from an expert or teacher. It may be given to the system interactively or *a priori*, and can take many different forms (see, for example [BUN90]).

[*] Submitted to *Journal of Experimental and Theoretical Artificial Intelligence*.

Once a learning system has been trained, it can use its knowledge to generalize on new situations. Generalization can be viewed as a limited form of reasoning since it uses existing knowledge to infer new knowledge. In inductive learning, the goal is often to learn classification rules for sets of objects. In the proper context, the rules can be viewed as defaults and the classifications as target conclusions. The reasoning power of such systems is then characterized as (one-step forward) default reasoning. When such default, or rule-based, reasoning is enhanced with similarity-based reasoning (when no default applies), the system can perform useful plausible inferences, that account for several important commonsense protocols (see [SUN92], Appendix A).

We propose an incremental learning algorithm (ILA) that augments inductive learning from examples with prior knowledge in the form of precepts (see Section 2.2), and supports many useful characteristics that can be exploited to combine learning and reasoning. We address in particular the following issues.

- Incremental learning
- Non-uniformity of the knowledge used in training (rules and examples)
- Inherent non-monotonicity
- Extension vs. intension
- Low-order polynomial complexity

Our contribution is the proposal of a system that attempts to combine inductive learning with prior knowledge and reasoning in a unified framework. The model described in [SUN92, SUN93] is a connectionist approach to implementing the form of reasoning discussed above (with forward chaining). However, this model only deals with representational issues, and not learning. ILA's reasoning power is more limited, but the system is able to learn from examples. ILA's inductive learning is similar to NGE [SAL91], but ILA uses hyperplanes rather than hyperrectangles as generalized exemplars. ILA also bears some resemblance with IBL [AHA91b], in the similarity measure used. However, ILA also incorporates prior knowledge and must provide mechanisms to support reasoning. Finally, if no rules are induced from the examples in learning, then ILA degenerates to a simplified form of MBR [STA86].

Section 2 introduces ILA, presents an example, and gives a comparison of ILA with other related work. Section 3 presents several simulation results on many real-world datasets. Section 4 discusses each of the above issues in detail, as well as some of the difficulties associated with ILA, such as the ordering of information in learning. Finally, section 5 concludes the chapter.

2. Incremental Learning Algorithm (ILA)

ILA is an incremental, inductive learning system that effectively combines examples drawn from experience with the world (i.e., a training set), with possible *a priori* knowledge, in the form of *precepts* obtained from some external source. We give a brief overview of ILA, introduce some preliminary definitions, show the details of ILA, and finally present a simple example. A brief comparison of ILA with other related models is also given.

2.1. ILA - Algorithmic Overview

ILA's execution mode implements a simple form of default (i.e., rule-based) reasoning, augmented with similarity-based reasoning. If a rule can be found that covers the new situation, then the rule is applied, otherwise, a "best match" is found between the new situation and data in the current knowledge base. Exceptions are kept and given priority, thus allowing ILA to account for inheritance. Conflicting defaults are handled by either static priorities, given *a priori* with the defaults, or by dynamic priorities obtained by counting exceptions to conflicting defaults.

ILA's learning consists of first executing the system, and then adapting the current knowledge base on the basis of the relationship (to be defined in Section 2.2) between the incoming training situation and its "best match." ILA may incorporate the new situation as is, use it to generalize its current knowledge, or update other information, such as priority counts.

ILA stores its knowledge base in the nodes of a network that is a balanced binary tree. Data are presented incrementally, and the system adapts by dynamically adding nodes to the network. The number of parameters is kept to a minimum, and the final network is the result of a sequence of self-organizing transformations. Both learning and execution require only simple gather and broadcast mechanisms. Hence, ILA makes use of inherent parallelism in execution, and exhibits low-order polynomial complexity.

Note that the use of a binary tree architecture is for purposes of consistency with previous models [GIR93b, GIR94b], and for the support of future extensions. For ILA, it is only necessary for the efficient identification of the node representing the best match with the training situation.

2.2. Preliminary Definitions

Let A be an application having inputs $i_1, ..., i_n$ and output o, with domains $I_1, ..., I_n, O$, respectively. Then $I=I_1\times...\times I_n$ is the *input space* and O is the *output space* for A. The training set is a subset of $I\times O$ from which A must be approximated. In this work, the approximation is a function from I to O.

A pair $p = (i, o)$ in $I\times O$ is called an *example*. Note that the domain of each input is assumed to contain the special symbol, ?, to account for possible missing attribute values in real-world examples. An example maps a single point of the input space to a value in O.

Precepts and rules are pairs in which some of the i-entries (inputs) are set to the special value *don't-care* (∗). The don't-care value has the following semantics. If i_j is an input whose value is don't-care in a pair p, then every pair p' obtained from p by asserting i_j to any one of its possible values (in I_j-{?}) and leaving p's other inputs unchanged has the same output as p. Informally, this says that in the context of p, the value of i_j has no effect on the value of the output. A pair containing don't-care inputs *represents* several examples. Thus, precepts and rules are more general than examples.

Precepts can be viewed as a result of some instantiation of domain knowledge or common sense. They can be given *a priori*, or at any time during learning. Precepts serve as learning biases, and are not necessarily correct. Thus, mechanisms exist to identify and deal with exceptions. Rules, on the other hand, are inductively synthesized from examples. By combining inductive learning and prior knowledge, ILA may be biased with precepts that are typically known to hold, or may naturally adapt to its environment. The former ability makes the system less sensitive to poor or atypical learning environments, while the latter increases the amount of adaptability, and makes the system viable in ill-defined environments (i.e., environments in which little is known *a priori* of the governing rules).

ILA induces rules by using the *dropping condition* generalization rule [MIC83a]. In the context of ILA's attribute-value representation language, it consists of setting to don't-care the value of one or more attributes in a pair. Only one attribute at a time may be set to don't-care. The decision of which attribute and when is made as follows. First, given two pairs x and y, we say that x and y are *concordant* if and only if they have the same output value, and x and y are *near_match* if and only if they differ in the value of exactly one non don't-care input. Then, if x and y are concordant and near_match, they can be *generalized* to a single rule obtained by setting to don't-care the attribute on which they differ, in one of the pairs. Alternative definitions of near_match give alternative conditions for the creation of generalizations.

The definition of near_match requires the ability to decide whether two input values are the same. This is clearly a simple task. However, with continuous attributes, the probability of two values being the same is essentially 0, and the above generalization mechanism becomes unsuitable. Indeed, no generalization could ever take place. Hence, the following modified definition of equality is used for deciding a near_match. For a nominal attribute, two values are *equal* if they are the same, and for a continuous attribute, two values are *equal* if they are within 20% of that attribute's range. Note that the choice of 20% is quite arbitrary and could potentially be replaced by a more appropriate (possibly learned) value, based say on the attribute's standard deviation. To facilitate generalization, two don't-know values are assumed to be equal.

The above definition of equality is extended to pairs in the natural way. Let x and y be any two pairs. We say that x is a *subset* of y if and only if all non don't-care attributes of x have values that are equal (in the above sense) in x and y. In the nominal case, x is a subset of y if and only if the set of points represented by x is a proper subset of the set of points represented by y. The modified definition of equality slightly extends this notion for the linear case. We say that x is a *superset* of y if and only if y is a subset of x. If x and y differ in the value of at least one non don't-care input, they are said to be *discriminated*. Finally, we say that *x covers y* if and only if x and y are concordant and y is a subset of x, and we define *num_asserted(x)* to be the number of non don't-care inputs in x.

Each node stores a pair, together with the pair's priority value (*pty*), counters for each output value (vector *counters*), and a counter for pairs covered (*num_covers*). A node's priority is a static value. It is 0 by default, but can be set *a priori* to any value greater than 0. These static priorities are useful for dealing intensionally with conflicting defaults. A node's counters are used for noise handling. Exactly one counter value is incremented each time an incoming training pair is equal to the node's stored pair. The value incremented corresponds to the training pair's output. The counter value that is highest represents the "most probable" output for the node's stored pair, and can thus be used as the node's output in execution. A node's num_covers value corresponds to the number of training examples and precepts that the node covers. It can serve as a dynamic priority for dealing extensionally with conflicting defaults, and is a measure of the level of confidence in the node's stored pair.

A node's activation is the "distance" of its stored pair's input to the input it is presented. The distance is defined as follows.

$$D(\boldsymbol{x},\boldsymbol{y}) = \frac{\displaystyle\sum_{i=1}^{n} \begin{cases} x_i \neq y_i & \text{if } x_i, y_i \neq *,? \text{ and attribute } i \text{ is nominal} \\ \dfrac{|x_i - y_i|}{range(i)} & \text{if } x_i, y_i \neq *,? \text{ and attribute } i \text{ is linear} \\ .5 & \text{if } (x_i = ?) \vee [(x_i \neq *) \wedge ((y_i = *) \vee (y_i = ?))] \\ 0 & \text{otherwise} \end{cases}}{num\_asserted(\boldsymbol{x})}$$

where *range(i)* is the range of values of attribute *i*

D is an extension of the heterogeneous distance measure proposed in [GIR94c]. A thorough treatment (motivation and justification) of some of the issues leading to the above definition is found in [GIR94c]. Note that *D* does not define a metric over the input space. However, it naturally accounts for the presence of don't-cares and for missing attributes in input vectors. Several alternatives for treating don't-know (or missing) attributes are in [AHA91a].

2.3. ILA - Algorithmic Details

ILA's learning consists of two phases: *execution* and *adaptation*. In the execution phase, the training pair is presented to the network and a winning node is identified. In the adaptation phase, counters and priorities are updated, and a new node may be created based on the relationship of the winning node's stored pair to the training pair. The algorithms given below are repeated for each new training pair *Q*. Each node stores some pair *P*.

Execution Phase

Broadcast(*Q*)
Gather((*DHET'*(*P*, *Q*), min), (num_asserted, min), (pty, min), (num_covers, min))

The details of the gather operation are given in the appendix. Essentially, it consists of having each node compute its activation, and as values are passed up the tree to the parent, a local winner is identified by first minimizing the activation (i.e., node whose stored pair is closest, or most similar wins), then resolving ties by giving priority to the more specific node (i.e., larger value of num_asserted), then to the node with highest priority, and finally to the node that has been matched most often so far (i.e., larger value of num_covers). Winning activations are passed up and nodes are tagged as self, right or left, based on which node wins the competition. An overall winning node is then easily identified.

Giving priority to more specific nodes helps in handling exceptions. Giving priority to nodes with highest static priority accounts for certain intensional conflicting defaults.

Finally, giving priority to nodes whose num_covers value is greatest increases the chances of selecting a "most likely to be correct" winner.

Let W be the winning node identified by the execution phase. If the network is empty, then W does not exist, and a new node is automatically added for the training pair. In all other cases, the following is performed. Let P be the pair stored in W, and Q be the current training pair.

```
Adaptation Phase
    diff = | num_asserted(P) - num_asserted(Q) |
    if (equal(P, Q)) then
      increment P.counters(Q.output)
      if (Q.pty > P.pty) then
        P.pty = Q.pty
        increment P.counters(Q.output)
      else if (P.pty > Q.pty)
        increment P.counters(P.output)
    else if (superset(P, Q) & concordant(P, Q)) then
      increment P.num_covers
    else if (discriminated(P, Q) & concordant(P, Q) & near_match(P, Q) & diff ≤ 1)
    then
      if (P.num_asserted ≤ Q.num_asserted & P.num_asserted > 1) then
        generalize P
      else if (Q.num_asserted > 1) then
        generalize Q
        create a node for Q
    else if (subset(P, Q) & concordant(P, Q)) then
      increment Q.num_covers
      delete W
      create a node for Q
    else
      create a node for Q
```

As nodes are added to the network, the binary tree structure is kept balanced. Generalization consists of setting to don't-care the value on which the two pairs disagree. The various conditions on when to generalize have the following purposes. The discriminated and concordant conditions are self-explanatory. The near_match condition guarantees that pairs that produce a generalization are, in some sense, close to each other in the input space. The condition on the relative number of asserted inputs (variable diff) controls the level (or sparsity) of generalization. A small value amounts to no generalization at all, while a large value may cause the creation of rules that are too general (i.e., the final representation is too sparse). The difference of at most 1 is supported by empirical evidence. The condition on relative num_asserted values increases the amount of generalization, and the condition on number of asserted inputs guarantees that no rule is created that has only don't-cares and thus covers the entire input space.

After all training pairs have been seen, ILA checks every node, and sets its output to the value whose counter is highest. This simple mechanism enables ILA to deal with noise. The inherent symmetry in the "equal" case, and between the "superset" and "subset" cases, is a reflection of ILA's attempt at reducing the impact of ordering on learning.

Evaluating ILA for predictive accuracy on a test set consists of presenting every pair in the test set and only running the execution phase. The output of the winning node is the system's output and prediction of the new pair's output.

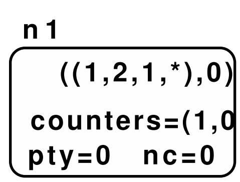
2.4. ILA - Example

We show a simple example that exercises ILA. Let the application have 4 nominal inputs, each with domain {0, 1, 2}, and a binary output. Let the set of training pairs consist of the following:

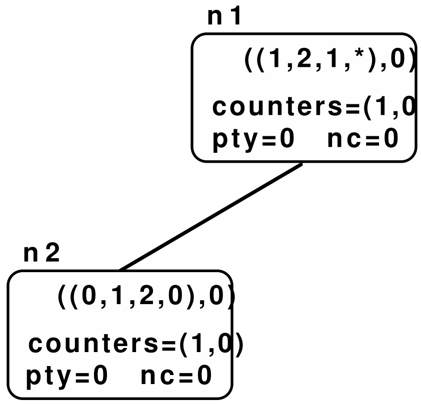
- $p_1 = ((1, 2, 1, *), 0)$ pty = 0
- $p_2 = ((0, 1, 2, 0), 0)$ pty = 0
- $p_3 = ((1, 2, 1, 2), 1)$ pty = 0
- $p_4 = ((1, 2, 1, 0), 0)$ pty = 0
- $p_5 = ((*, 0, 0, 1), 1)$ pty = 2
- $p_6 = ((1, 2, 0, 2), 1)$ pty = 0
- $p_7 = ((0, 1, *, *), 1)$ pty = 1

The network starts empty and each pair is incrementally presented to it. Note that p_1, p_5, and p_7 are precepts. We show the evolution of the network after each presentation. We use nc as a short-hand for num_covers, *n.p* to denote the pair stored in node *n*, and D(*n*) for *D*(*n.p*, current training pair).

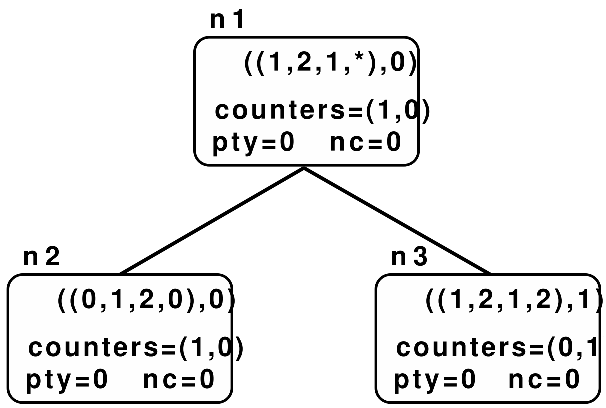
<u>Presentation of $p_1 = ((1, 2, 1, *), 0)$</u>. Since the network is empty, there is no winner and a new node n_1 is created for p_1.



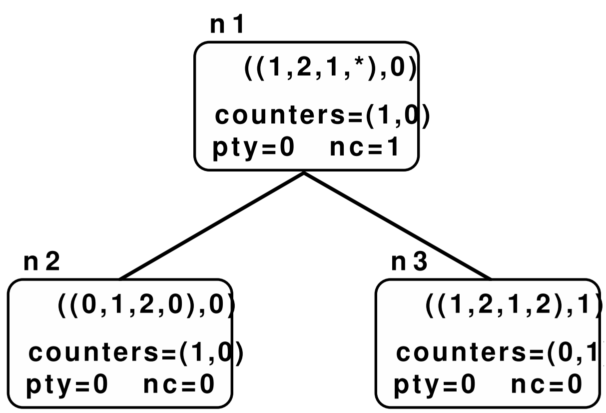
<u>Presentation of $p_2 = ((0, 1, 2, 0), 0)$</u>. The execution phase marks n_1 as the winner with $D(n_1)=(1+1+1+0)/3=3/3=1$. Pairs $n_1.p$ and p_2 are discriminated and concordant, but not near_match. No generalization can take place, and a new node n_2, is created for p_2.



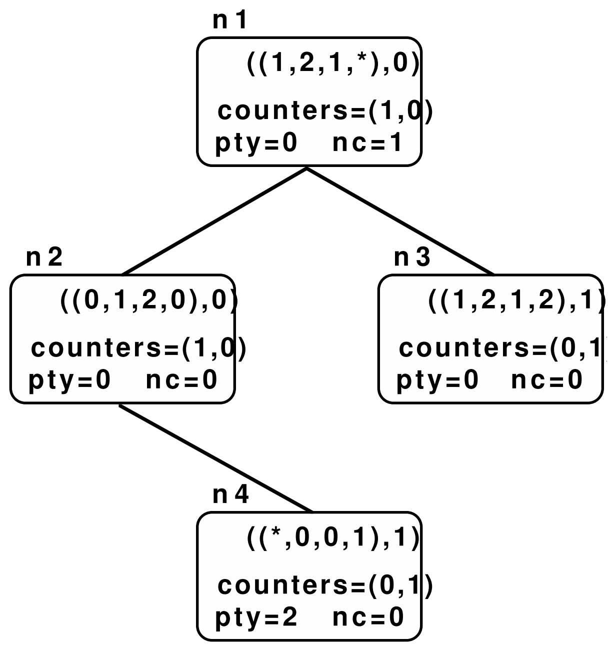
<u>Presentation of p_3 = ((1, 2, 1, 2), 1)</u>. The execution phase marks n_1 as the winner with D(n_1)=0, and D(n_2)=1. Pair $n_1.p$ is a superset of p_3 and discordant (i.e., p_3 is an exception to $n_1.p$). Hence, a new node n_3, is created for p_3.



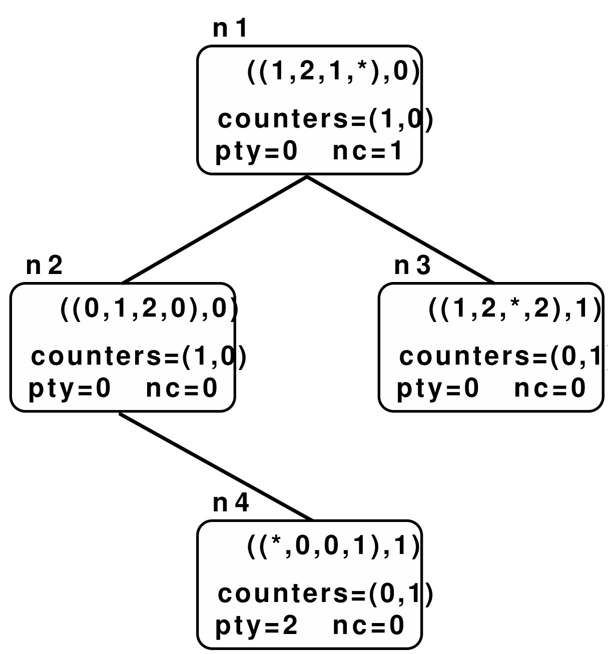
<u>Presentation of p_4 = ((1, 2, 1, 0), 0)</u>. The execution phase marks n_1 as the winner with D(n_1)=0, D(n_2)=3/4, and D(n_3)=1/4. Pair $n_1.p$ is a superset of p_4 and concordant. Hence, num_covers(n_1) is incremented, and no new node need be created.



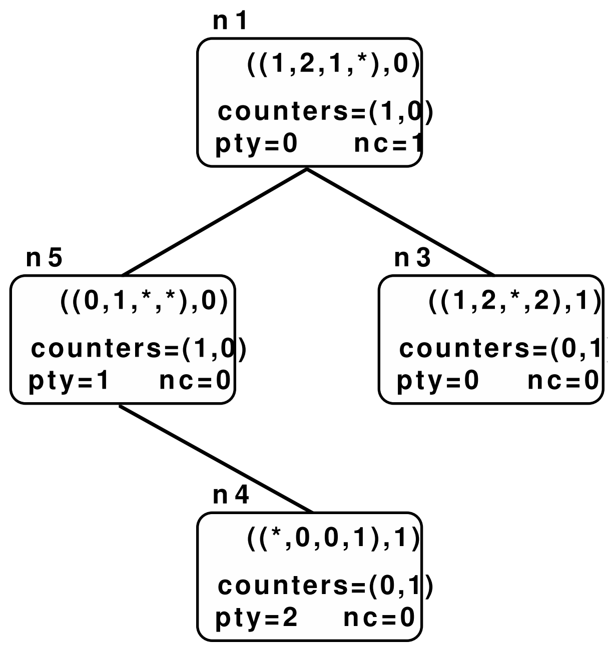
Presentation of p_5 = ((*, 0, 0, 1), 1). The execution phase marks n_1 as the winner with $D(n_1)=5/6$, $D(n_2)=7/8$, and $D(n_3)=7/8$. Pairs $n_1.p$ and p_5 are discriminated and discordant. No generalization can take place, and a new node n_4, is created for p_5.



Presentation of p_6 = ((1, 2, 0, 2), 1). The execution phase marks n_3 as the winner with $D(n_1)=1/3$, $D(n_2)=1$, $D(n_3)=1/4$, and $D(n_4)=1$. Pairs $n_3.p$ and p_6 are discriminated, concordant, near_match, and diff=0≤1. So generalization takes place. Their differing input is replaced by the special don't-care value in $n_3.p$, and no new node need be created for p_6.



Presentation of p_7 = ((0, 1, *, *), 0). The execution phase marks n_2 as the winner with D(n_1)=5/6, D(n_2)=1/4, D(n_3)=5/6, and D(n_4)=2/3. Pair $n_2.p$ is a subset of p_7 and concordant. So n_2 is deleted and a new node n_5, is created for p_7. The network structure remains unchanged.



In this simple noise-free, non-redundant example, the counters are never changed. As discussed earlier, the counters are useful in noisy situations. For example, assume the above training set is augmented by the following two additional training pairs.

- p_8 = ((*, 0, 0, 1), 0) pty = 2
- p_9 = ((*, 0, 0, 1), 0) pty = 2

Then the final network remains the same, but counters(n_4) becomes (2, 1), and so the output of n_4 is set to 0 rather than 1.

At any time, the network can be used to predict the output of new patterns. In such cases, only the execution phase is performed, and the output of the pair stored in the winning node becomes the output of the network for the new pattern. Consider, for example, the network after p_1 through p_7 have been seen. Suppose, we want to predict the outputs of the following two inputs:

- i_1 = (1, 2, 1, 2)
- i_2 = (?, 1, ?, ?)

In the case of i_1 ,we have D(n_1)=D(n_3)=0, and D(n_5)=D(n_4)=1. Since n_1 and n_3 have the same priority, but num_covers(n_1)>num_covers(n_3), n_1 wins and the output for i_1 is 0. In the case of i_2, we want the default output (or classification) of patterns whose second attribute is asserted to 1. We have D(n_1)=D(n_3)=D(n_4)=2/3, and D(n_5)=1/4, so that n_5 wins and the output for i_2 is 0.

Note that if the output of some of the new patterns were known, then ILA could continue to learn from them over time by simply executing the learning phase.

2.5. Comparison with Other Work

ILA emerged as a result of extensions to PDL2 [GIR94b]. ILA does preserve the combination of rule-based and similarity-based reasoning mechanisms of PDL2. In ILA however, there is a closer match between the learning and the execution algorithms, and modifications to the knowledge base are confined to the winning node rather than extended to all nodes.

Because changes are localized to the winning node, much of PDL2's complex exception handling mechanisms can be done away with. Indeed, since execution gives priority to more specific nodes and the activation of a node is based on similarity with the new input, linear inheritance is automatically accounted for. Moreover, because ILA always starts with an execution phase that identifies a winner to make changes with, it is easier to use the current network's prediction on the training pair to make the system even more incremental. The system can indeed decide that if the winning node's stored pair is concordant with the training pair, and they are "close enough" to each other, then no modifications need be made to the network. The value of the winning activation can be used to vary the degree of closeness. Results of simulation with one possible value are reported in Section 3.

ILA uses a heterogeneous distance measure, while PDL2 uses homogeneous, purely nominal distance measure. It follows that ILA can deal with continuous inputs directly, without having to first discretize them. Moreover, ILA's measure preserves the notion of order while PDL2's does not, thus resulting in ILA's improved performance on datasets containing ordered data (see [GIR94c]). ILA and PDL2 exhibit similar performance in actual learning time, but ILA results in better predictive accuracy on average.

ILA bears some similarity with NGE [SAL91] as well. However, because ILA generalizes only by setting some attribute(s) to don't-care, the produced generalizations, or generalized exemplars [SAL91], are hyperplanes, rather than hyperrectangles, in the input space. Hence, ILA can be viewed as a nearest-hyperplane learning algorithm. Many of the mechanisms are similar, but ILA also allows static and dynamic priorities to break ties in equidistant generalizations. Moreover, where it was shown that overlapping hyperrectangles may hinder performance [WET93], ILA allows overlapping hyperplanes for purposes of dealing with conflicting defaults.

In the case that no generalizations are found, ILA degenerates into a restricted form of MBR [STA86]. This allows ILA to perform well, even with simple (and possibly weak) generalization mechanisms (see for example the soybean-small dataset in Section 3). Going further, one may consider extending ILA with some of the well established mechanisms of MBR to further improve ILA's performance in situations where only few rules are available.

The distance used by ILA is very similar to IBL's metric [AHA91b]. ILA's distance measure also deals with don't-care attributes (which are non-existent in instance-based learners), and treats missing attributes somewhat differently. Where IBL considers them to be complete mismatch, ILA chooses a more middle-ground approach that may better capture the inherent notion of missing (or "don't-know") attributes.

Finally, ILA's combination of default (i.e., rule-based) reasoning and similarity-based reasoning follows CONSYDERR [SUN92]. However, CONSYDERR is strictly concerned with a connectionist approach to concept representation and commonsense reasoning. The resulting model is elegant, but it does not address the problem of learning.

3. Simulation Results

ILA was tested on several datasets, both to exercise its learning potential (i.e., its ability to inductively synthesize rules from examples to produce high predictive accuracy on some test set), and to test its ability, through incrementality, to deal with important commonsense protocols, such as inheritance, conflicting defaults, missing information, and the lack of a matching rule.

3.1. Inductive Learning and Predictive Accuracy

Several datasets from [MUR92] were chosen. The selected applications and their essential attributes are given in Table 1. They represent a wide variety of situations.

Table 1 - Selected Applications and Attributes

Dataset	#instances	#inputs	Input space	#out. values
zoo	90	16	Nominal	7
iris	150	4	Linear	3
lenses	24	4	Lin/Nom	3
hepatitis	155	19	Linear	2
voting-84	435	16	Boolean	2
glass	214	9	Linear	7
breast-cancer	699	9	Nominal	2
pima-indians	768	8	Linear	2
vowel	528	10	Linear	11
LED	1000	7	Boolean	10
soybean-small	47	35	Lin/Nom	4

ILA's results were gathered for each of the above applications. They are shown in Table 2. Each of the reported results is an average over 10 runs of ILA. In each run, the training set and the test set are randomly regenerated. Each set contains one half of the examples in the complete data set. The first number (PA) represents predictive accuracy (in %) on the test set after training and the second number (GR) is the ratio of the size of the final network to the number of instances used in learning. It is another measure of the generalization power of ILA, as well as an indication of ILA's memory requirements. The results given here were obtained with no precepts given a priori. That is, the rules used in execution/reasoning (reflected in PA) are all created during learning.

Table 2 - ILA Simulation Results

Application	PA	GR
zoo	95.3	.47
iris	95.1	.20
lenses	75.8	.50
hepatitis	80.8	.97
voting-84	91.6	.68
glass	58.9	.27
breast-cancer	95.8	.48
pima-indians	70.3	.35
vowel	76.1	.32
LED	70.5	.15
soybean-small	99.1	1
AVERAGE	82.7	.49

These results compare favorably with results reported for other inductive learning algorithms (see [ZAR94] for a summary). Note that LED is a noisy application, for which the Bayesian optimal is 74%. ILA handles the noise well, both by achieving high predictive accuracy, and by constructing a significantly small network.

Simulations of four of the applications (zoo, lenses, hepatitis, voting-84) with precepts given *a priori* (before the training set is presented) were also conducted. Results are reported in Table 3. Each column shows both PA and GR.

Table 3 - ILA with Precepts

Application	w/o precepts	w/precepts
zoo	95.3 - .47	92.4 - .42
voting-84	91.6 - .69	95.1 - .28
lenses	75.8 - .50	80.8 - .42
hepatitis	80.8 - .97	82.2 - .66

The results with precepts show an increase of 2.3% in predictive accuracy, and a decrease of 29.5% of GR (i.e., the resulting networks are significantly smaller on average). The precepts for each dataset are as follows.

<u>zoo dataset:</u>

1. If animal has four legs, then animal belongs to class 1
2. If animal has feathers, then animal belongs to class 2
3. If animal lays eggs, is aquatic, and has fins, then animal belongs to class 4

<u>lenses dataset:</u>

1. If patient has low tear production rate, then patient is not fit for contact lenses

<u>voting-84 dataset:</u>

1. If representative voted 'no' on the 'physician-fee-freeze' issue, then rep. is a democrat

<u>hepatitis dataset:</u>

1. If patient is between 21 and 30, then patient lives

2. If patient is between 51 and 60, is a male, uses steroids, has malaise, has a liver that is big and firm, has high bilirubin and high albumin, then patient dies

Currently, the precepts are obtained from domain knowledge provided with the application (voting-84) or generated from the authors' common sense (zoo, lenses, hepatitis). Other, more automatic ways of generating precepts are the topic of future research.

Because ILA's learning starts with an execution phase that identifies a best match, ILA can easily be modified to increase speed and parsimony at the possible expense of predictive accuracy. Rather than always going through the adaptation phase, ILA first checks the output of the winning node and the distance of that node's stored pair to the training pair. If the winning node's output is identical to the training pair's output and its distance is below a certain threshold *T* (i.e., close enough to the training pair), then the adaptation phase is skipped and the next training pair is presented. The threshold *T* then becomes a parameter whose value can be fine-tuned to the user's goal. Averages for PA and GR over the eleven selected datasets with *T*=2 are given in Table 4. Results for the basic ILA model and PDL2 are also included for comparison.

Table 4 - Comparative Study

Algorithm	PA	GR
ILA	82.7	.49
ILA, *T*=2	73.9	.20
PDL2	79.7	.66

As expected, results with *T*=2 show a decrease in PA (about 10%), but also a significant decrease in GR (over 59%). For three of the datasets (zoo, breast-cancer, and soybean-small), the decrease in PA is less than 1.1% on average, while the decrease in GR is greater than 76%. The threshold *T*, though not part of the basic model, provides direct and simple control of the PA/GR trade-off. Also, ILA's results exceed PDL2's.

3.2. Incremental Learning of Reasoning Patterns

ILA allows the incremental learning of default rules, either extensionally (i.e., using induction from examples) or intensionally (i.e., using precepts given *a priori*). Once the rules have been learned, the system can be used to reason about them. Essentially, ILA's execution combines a form of default reasoning with similarity-based reasoning, as in [SUN92]. This combination decreases the system's susceptibility to brittleness. It has also been shown to effectively account for many commonsense protocols, such as inheritance (see [SUN92], Appendix A).

Several problems, from the benchmark of nonmonotonic reasoning problems described in [LIF88], were presented to ILA. Because of their symbolic nature, the problems are first translated into their corresponding attribute-value language representation, where each predicate gives rise to a Boolean attribute, each class of objects gives rise to a multi-valued attribute, and each implication (or default) gives rise to a precept (see [GIR94b] for details). ILA is able to properly incorporate the premises, and correctly derive the expected conclusions for the following classes of problems:

- A1 - basic default reasoning
- A2 - default reasoning with irrelevant information
- A5 - default reasoning in an open domain
- A9 - priority between defaults
- B1 - linear inheritance (top-down)
- B2 - tree-structured inheritance
- B3 - one-step multiple inheritance

Because ILA only handles single output applications, it cannot deal with default reasoning with several defaults, multiple inheritance, or situations that require forward chaining. Essentially, it is limited to "one-step" forward plausible inferences. Also, there is no notion of time in ILA, so that reasoning about actions is not possible.

ILA was also tested on its extensional abilities on two particular problems. The first is the Nixon diamond [REI81]. ILA solves it both intensionally (i.e., default rules and priorities given *a priori*), and extensionally from a set of examples. Examples correspond to instances of Republicans that are not Quakers, Quakers that are not Republicans, and Republicans that are also Quakers, together with the output they imply, that is, not pacifist, pacifist, and either, respectively. The relative number of instances with a given output value in the latter category is used to get a dynamic value of the relative priorities of the default rules. Note that the order of the instances may affect the outcome.

The second problem was constructed to further demonstrate the adaptability of ILA to its environment in a purely extensional situation. It serves as a simple, preliminary example of the combination of incremental inductive learning with reasoning. It consists of the classical problem of birds, penguins, and their ability to fly. The application has four input attributes, two of which correspond to Penguin and Bird. The other two are other (undetermined) attributes of birds (e.g., Feather). The output is Boolean and characterizes the ability to fly. At first, the system is exposed mostly to penguin-birds. When asked whether birds typically fly (i.e., only the Bird attribute is asserted, and all other inputs are don't-know), ILA concludes that birds do not fly. This response is consistent with ILA's current experience with the "world." However, as more new instances of flying birds (i.e., other than penguins) are encountered, ILA adapts its knowledge and when asked again, concludes that birds fly. This simple case of inheritance can thus be handled directly (B1), or extensionally.

4. Characteristics of ILA

In this section, we examine several of ILA's characteristics, relevant to the combination of learning and reasoning. We also discuss the problem of ordering that is inherent to one-shot learners.

4.1. Incremental Learning

We first define what is meant by incrementality and then discuss the validity of incremental learning systems. We say that a learning system is *incremental* if the following conditions hold:

1. Training examples need not be available *a priori* in a training set, but can become available one at a time over time (although many incremental systems still use a training set in practice),
2. Only minor revisions are made to the current knowledge base as each new training example is presented,
3. No major reprocessing of previous examples is ever performed, and
4. Learning may go on continually (as long as examples are presented to the system).

Classical training set learning is typically based on the assumption that the training set contains many examples representative of the application to be learned. In other words, inductive mechanisms will eventually extract from the training set the necessary critical features. This approach typically assumes that enough information is available *a priori* and learning is complete once convergence has been achieved on the training set. Humans however acquire knowledge over time, or incrementally. All the information necessary to learn a given concept is rarely available *a priori*. Rather, new pieces of information become available over time, and the knowledge base is constantly revised (i.e., evolves) based on the newly acquired information. In the context of learning grammar with a recurrent network, empirical evidence is given in [ELM91] that the "network fails to learn the task when the entire data set is presented all at once, but succeeds when the data are presented incrementally [with an easy to hard ordering]." Humans never cease to learn, thus minimizing the impact of poor or atypical, temporary learning environments (i.e., "bad" training sets).

Also, if only minor revisions are made to the current knowledge base as new information is gained, and no major reprocessing of previous observations need be done, then an incremental learning system may allow a reduction of overall algorithmic complexity. Moreover, in the context of the backpropagation algorithm, evidence has been given that better results are achieved with on-line (or incremental, after each training example) update of the weights than with batch update [LEC89].

Since learning takes place over time and no training set need be generated *a priori*, incremental learning systems may be left to naturally adapt to their environment. In nature, incrementality allows the natural filtering out of noisy examples. Spurious or noisy examples, after they have been "seen" once, will either not reoccur at all, or are likely to reoccur without noise. Hence, incrementality provides natural mechanisms to handle noise.

4.2. Non-Uniformity of Knowledge

Seldom do humans learn exclusively from random examples, without ever being exposed to more general (commonsense) rules. Rather, because of built-in mechanisms (e.g., pain) and social structures (e.g., the family, school), humans are able to combine both specific examples and more general rules to efficiently learn complex problems. For example, regardless of most external circumstances, it does not make sense to put your hand on a red-hot stove. This is a general rule or precept that does not require successive examples of settings involving your hand on the stove, to be acquired. Instead, exposure to precepts taught by others (e.g., "it will burn you"), together with a built-in pain mechanism, allow the (almost instantaneous) learning of the more general rule. Foreign-language students first learn English rules (e.g., the plural of a noun is formed by tagging an *s* to the end of the singular form of that noun), and are then exposed to exceptions (e.g., geese instead of gooses, mice instead of mouses, etc.). It is suggested in [ELM91] that this non-uniformity of knowledge in learning may actually be the result of an early handicap (e.g., limited memory and attention span characteristic of children). In other words, it may be that the learning system itself evolves over time, rather than the knowledge it is presented.

Whatever the case, the ability to use rules and examples increases learning speed by pruning and constraining the search for generalizations in the input space, reduces memory requirements, and improves overall predictive accuracy (see, for example [GIR93b]). In practical situations, the rules can be viewed as some instantiation of domain knowledge available *a priori* and given to the system. They serve as learning biases that supplement the inductive mechanisms to increase the probability that known rules are available to the system in reasoning. As such rules can be given *a priori*, the system does not have to suffer from poor or atypical environments, which is the "flaw" of systems based solely on inductive learning from examples (see Section 4.5).

4.3. Chronology as a Learning Bias

Chronology (i.e., the order in which knowledge is acquired) is an inherent aspect of incrementality. Thus it is possible to implicitly integrate time as a factor and another source of bias in learning. For example, the system could choose to give precedence to newly acquired knowledge, recognizing the possibility that early guesses may be incorrect (see, for example [MAR94]). In an iterated training set system, the distribution of "weight" over the various pieces of knowledge is uniform. So, a temporal ordering of the various pieces of knowledge would have to be made explicit (most likely by some external teacher) to achieve the same result.

A learning system's experience with the world could be appropriately ordered (e.g., general rules before exceptions) so as to increase its efficiency. This "ordering" is reminiscent to the way humans deal with their environment. A small set of default rules allow many problems to be solved and form the basis of effective default reasoning. Only when the default rules seem to be invalid, do we need to adapt to the situation (i.e., learn the exception).

4.4. Non-Monotonicity

Classical first-order logic is monotonic. Informally, this says that the acquisition of new knowledge can never invalidate previously known facts. Though theoretically convenient, monotonicity certainly does not hold in everyday life, where information is often uncertain and incomplete. Consider the following classical example. Suppose someone tells you that Tweety is a bird and asks you whether Tweety can fly. Since you know that birds typically fly, and you have no reason to believe otherwise concerning Tweety, you are most likely to say that Tweety can fly. Now, imagine that later on you are told that Tweety is a penguin. Then, since you know that penguins, though they are birds, do not fly, you will withdraw your previous conclusion and say that Tweety cannot fly.

The new piece of knowledge you gained invalidated the truth of the previously accepted fact.

Such patterns of reasoning, often equated with commonsense reasoning by logicians (see, for example [BRE91, LUK90]), are pervasive in the way humans deal with the inherent uncertainty and incompleteness of the world (or their representation thereof). The development of systems that effectively handle classical commonsense protocols, such as inheritance, is a first step in overcoming the brittleness bottleneck.

Non-monotonicity is a consequence of incrementality. If all the knowledge is available *a priori*, then non-monotonicity is not an issue. That is, there is no need for special learning mechanisms that invalidate portions of knowledge, while not affecting the rest of it. Essentially, the world is closed. If knowledge is incrementally made available, however, then more work is required. Consider the above example again. Knowing that Tweety is a penguin should only directly and explicitly affect knowledge about Tweety, not any other birds. Just because I now know that Tweety is a penguin and that penguins do not fly, I (most likely) would not want to conclude that Fred (another bird that I encounter later) does not fly, even though my belief in Fred's flying ability may be (implicitly) lessened by my having discovered the existence of exceptions to the rule.

4.5. Extension vs. Intension

We have seen that rules can be used as biases to supplement inductive mechanisms. It is also true that with appropriate generalization mechanisms, examples can effectively supplement rules given *a priori* by generating new rules, and modifying existing ones. Hence, the incremental system combines the intensional approach (based on features, expressed by rules) and the extensional approach (based on instances, expressed by examples). Such a combination may prove useful in commonsense reasoning. In particular, it allows the system to retain the self-adaptivity inherent in human learning, while not having to unnecessarily suffer from poor or atypical learning environments.

One useful application of the above is the case of conflicting defaults, such as the famous Nixon Diamond [REI81]. If the extensional and intensional approaches are combined, greater flexibility can be achieved. There are at least three available options: 1) default rules are given *a priori*, along with their respective priority (e.g., religion is stronger than politics), 2) default rules are given *a priori*, but their relative priorities are computed dynamically via exposition to a set of examples, or 3) default rules and their priorities are inductively learned by the system. The first option guarantees the expected result (according to some teacher), the second option adds flexibility by letting experience supplement *a priori* knowledge, and the third option produces a model consistent with current experience with the world. These options represent a trade-off between self-adaptivity of the system and consistency of the system with some external entity's representation of the world.

If the goal of the system is to emulate human reasoning processes, and domain knowledge can easily be captured (by some external teacher) and given to the system, then the intensional approach is to be preferred. However, if little is known of the domain of application, or the goal is to design a system capable to adapt to and evolve in its current environment, then the extensional approach must be used. We suggest that the ability to use either and to combine them offers the maximum flexibility. This combination has also been recognized as necessary in the design of practical embedded (autonomous) agents [KAE93].

4.6. Low-Order Polynomial Complexity

ILA has low order polynomial complexity in the number of training pairs. First, define:

- n, the number of inputs of the application,
- m, the number of output values of the application,
- t, the number of training pairs, and
- N, the number of nodes in the final network.

Then broadcast is at worst $O(nt\log N)$, gather is at worst $O(nt+t\log N)$, and adaptation, ignoring the cost of balancing the tree, takes at most $O(tn)$ (when generalization takes place). The final phase that sets each node's output value is at most $O(m)$.

Now, for all practical purposes, n and m can be considered to be bounded above by some constant. Moreover, N is at most equal to t. Hence, ILA is essentially $O(t\log t)$.

4.7. Ordering

Though chronology may be used as a bias, it presupposes that the ordering of the data carries some meaning that should be implicitly captured by the learning system. Such is not always the case. Indeed, one may wish to have the flexibility of entering rules at any time during learning and expect the same outcome, regardless of the chosen ordering. This is a complex problem. It is partially solved by PDL2 [GIR94b] and ILA, as can be seen in the symmetry inherent to the algorithms. However, the systems remains sensitive to the ordering.

One can argue that for large (and rich) enough training sets, the effects of ordering become more limited, as the current representation of the system eventually matches reality. In other words, global statistical patterns ultimately override errors or misrepresentation due to local patterns of noise, lack of information, or plain errors. Indeed, incremental (on-line over time) learning presupposes redundancy in the data, as is true in human learning. Similar situations have a tendency to reproduce themselves so that general rules (or analogies) can ultimately be drawn from them.

5. Conclusion

In this chapter, we present an incremental learning algorithm (ILA) that attempts to combine inductive learning from examples with prior knowledge and reasoning. ILA is essentially a nearest hyperplane learning model. It incorporates prior knowledge in the form of precepts that are hyperplanes whose semantics are that of default rules. Finally, ILA implements a simple form of default reasoning, combined with similarity-based reasoning, thus allowing it to perform many plausible inferences. Results of simulation of ILA on many real-world datasets and commonsense protocols are presented and discussed. They demonstrate promise.

In an effort to cover breadth, ILA tends to suffer from lack of depth in all of the aspects it attempts to combine. Efforts are ongoing to improve generalization mechanisms in learning, to discover other forms of prior knowledge, and to enhance reasoning power. Future work must focus on extensions to handle multiple outputs and forward chaining. The critical question of the ordering of the training data remains open. In some cases, order is irrelevant, or at least should be treated as such, whereas in other cases, the ordering (or chronology) may carry meaningful information. More examples (possibly in the spirit of the UCI repository of machine learning databases) of applications requiring (or likely to benefit from) the combination of inductive learning with prior knowledge and reasoning should also be developed.

Appendix

In this appendix, we give the details of the gather operation of the execution phase of ILA. The gather operation has the following general syntax.

gather((value$_1$, flag$_1$), [(value$_2$ flag$_2$), ...])

The first pair of parameters is mandatory, and can be supplemented by any number of optional pairs of parameters. The semantics associated with each pair of parameters are as follows.

- value$_i$: a value to be computed at each node. Value$_1$ is the only one actually being gathered. In general, it corresponds to the node's activation.
- flag$_i$: a Boolean flag that determines whether value$_i$ is to be minimized or maximized

Gathering then proceeds as follows. Each node computes value$_1$. Since they have no children, the leaf nodes mark themselves as self, meaning that they win the competition. They then pass value$_1$ to their parent nodes. The parent nodes recursively do the following:

1. Compare their value$_1$ with that of their children
2. Determine a winner based on (flag$_1$, value$_1$). If a tie exists, the other specified pairs of parameters are used to try to break it. A tie on value$_i$ is broken with (value$_{i+1}$, flag$_{i+1}$). If the tie cannot be broken with (or before) the last value and flag, then the parent node is arbitrarily chosen as the winner.
3. Mark themselves as *self* if they win the competition, and as *right* (resp. *left*) if the right child (resp. the left child) wins the competition.
4. Send the winning value$_1$ up to their parent node

It is clear that the value$_1$ that reaches the top node is the one that best satisfies the ordered list of conditions expressed by all of the parameter pairs, over all the nodes. Also, a trace has been left from the winning node up to the top node. It is therefore trivial to go back down the tree by following the directions indicated by the mark of each node, until a node marked as *self* is encountered. Giving priority to the parent rather than the children in tie-breaking (step 2 above) makes the system less non-deterministic, but tends to minimize the depth of the path.

CHAPTER 8
An Integrated Framework for Learning and Reasoning*

ILA exhibits reasonable inductive learning capabilities that compare favorably with other existing learning algorithms. However, its reasoning scheme is very limited in that it does not support any kind of chaining. Also, the empirical studies conducted with ILA (and PDL2) are limited to "toy" problems. To better assess the validity of the models we have introduced in this dissertation, we must develop one or more "real-world" benchmarks. In this chapter, we first present an extended version of ILA, called FLARE (Framework for Learning and REasoning) that accounts for forward chaining and that can be instantiated both at the conceptual and subconceptual levels. We also introduce several more real-world examples combining learning and reasoning, and analyze the extended model on it. We also give a preliminary example of how reasoning instantiates "commonsense" knowledge to specific domains so as to produce precepts that can then be used as learning biases in inductive learning.

This chapter constitutes the culmination of the research presented throughout the dissertation. A better characterization of many important concepts (e.g., generalization, conflicting defaults) is given, a mechanical way of translating first-order statements into attribute-value vectors is proposed, and many yet unsolved (or ill-addressed) issues are thoroughly discussed. Several important areas of future research are identified. They are discussed further in Chapter 9.

1. Introduction

Induction and deduction both appear to be underlying processes in intelligent agents[2]. Induction "involves intellectual leaps from the particular to the general." [DIG84, p.13]. It is often considered to be a mechanism for knowledge acquisition or learning. Indeed, "All the laws of nature were discovered by inductive reasoning." [DIG84, p.13]. Deduction is a form of reasoning with and about acquired knowledge. It typically does not result in the generation of new facts, rather it establishes cause-effect relationships between existing facts. Deduction may be applied *forward* by seeking the consequences of certain existing hypotheses, or *backward* to discover the necessary conditions for the achievement of certain goals. Despite their differences, induction and deduction are strongly interrelated. The ability to reason about a domain of knowledge is often based on rules about that domain, that must be acquired somehow; and the ability to reason can often guide the acquisition of new knowledge, or learning.

Inductive learning has been the subject of much research leading to the design of a variety of algorithms [CLP89, MIC83a, QUI86, SAL91]. In general, inductive learning

---

* Submitted to *Journal of Artificial Intelligence Research*.

[2]The philosopher Pierce actually identified three basic forms of reasoning: inductive, deductive and abductive. See Hartshorn et al. (Eds.), *Collected Papers of Charles Sanders Pierce*, Harvard University Press, Vol. 2, pp. 1931-1958. We do not address abductive mechanisms here.

systems generate classification rules from examples. Typically, the system is first presented a set of examples (objects, situations, etc.), also known as a *training set*. Examples are usually expressed in the attribute-value language, and represent recorded instances of attribute-value pairs together with their corresponding classification. The system's goal is then to discover sets of sufficient critical features or rules, that properly classify the examples of the training set (convergence), and adequately extend to previously unseen examples (generalization).

Though machines are still a far cry from matching human qualitative inductive leaps, inductive learning systems have proven useful over a wide range of applications in medicine (breast cancer, hepatitis detection), banking (credit screening), defense (mine-rock discrimination), botany (iris variety detection, mushroom identification), and others [MUR92].

The study of deductive reasoning goes at least as far back as the early Greek philosophers, such as Socrates and Aristotle. Its formalization has given rise to a variety of logics, from propositional to first-order predicate logic to default logic to several non-monotonic extensions. Many of these logics have been successfully implemented in artificial systems (e.g., PROLOG, expert systems). They typically consist of a pre-encoded knowledge or rule base, a given set of facts (identified as either causes or consequences), and some *inference engine*. The inference engine carries out the deductive process using the rules in the rule base and the facts it is provided. Several of these systems have been successfully used in various domains, such as medical diagnosis [CLB84] and geology [DUD84].

One of the greatest challenges of current deductive systems is knowledge *elicitation*, that is the construction of the rule base. Typically, rules are obtained from human experts. Elicitation is a tedious task that presents many difficulties both practically and theoretically. If a sufficiently rich training set can be obtained, then inductive learning may provide an alternative. It is typically easier and faster to record examples than it is to gather rules. On the other hand, if some *commonsense* knowledge can easily be obtained and instantiated to the specific domain, then deductive reasoning may also be used to improve inductive learning by, for example, generating useful learning biases or *precepts* (see [GIR93b] for a definition). Note that rules and examples need not be mutually exclusive. A system's knowledge base can indeed be constructed from both prior rules and inductively generated ones. The strong knowledge principle [WAT86] and early work on bias [MIT80] suggest the need for prior knowledge.

Although proposals have been made to enhance rule-based (i.e., deductive) systems with some learning capabilities [HAA83, RYC83], we are not aware of any work attempting to combine induction and deduction into a unified framework. Yet, as indicated above these two processes share many interdependencies, and their integration may lead to more powerful models. This paper describes a system called FLARE (Framework for Learning And REasoning), that attempts to integrate inductive learning with prior knowledge and reasoning. In particular, inductive learning is handled incrementally, prior knowledge is used to prune and constrain the search in the input space, and reasoning is non-monotonic. Prior knowledge may be given by a teacher or deductively obtained by instantiating existing commonsense knowledge. Results on several datasets and classical commonsense protocols demonstrate promise.

The paper is organized as follows. Section 2 discusses related work in induction and deduction. Section 3 presents FLARE and further argues the validity of the unified framework. Section 4 shows simulation results on classical datasets and a benchmark of "well-designed" commonsense protocols. Section 5 introduces several preliminary examples of the use of the framework on more real-world applications. In particular, it shows reasoning patterns following learning, and automatic generation and use of prior knowledge. Finally, section 6 concludes the paper by summarizing the results, and discussing further research.

2. Related Work

FLARE follows in the tradition of [GIR94b, GIR94d], as it combines inductive learning with prior knowledge and reasoning. There does not seem to exist, to the authors knowledge, any other such attempt. There are however a variety of inductive learning models, reasoning systems, and models combining induction with prior knowledge, that bear similarity to the corresponding components of FLARE.

Induction in FLARE is carried out much the same way as in NGE [SAL91]. However, because generalization is effected only by setting some attribute(s) to don't-care, the produced generalizations, or generalized exemplars [SAL91], are hyperplanes, rather than hyperrectangles, in the input space. Hence, FLARE implements a nearest-hyperplane learning algorithm. FLARE also uses static and dynamic priorities to break ties between equidistant generalizations. Moreover, where it was shown that overlapping hyperrectangles may hinder performance [WET94], FLARE allows overlapping hyperplanes for purposes of dealing with conflicting defaults.

In the case that no generalizations are constructed from the training examples, FLARE degenerates into a restricted form of MBR [STA86]. The distance metric used is similar to IBL's metric [AHA91], but it also deals with don't-care attributes (which are non-existent in instance-based learners), and treats missing attributes somewhat differently. Where IBL considers them to be complete mismatches, FLARE chooses a more middle-ground approach that may better capture the inherent notion of missing or "don't-know" attributes.

Learning in FLARE contrasts with algorithms such as CN2 [CLP89], where all training examples must be available *a priori*. Rather, it follows an incremental approach similar to that of [ELM91], except that it is the knowledge itself that is evolved, rather than the system's structure. Moreover, learning in FLARE can be effected continually.

Systems combining inductive learning with prior knowledge include PDLA [GIR93b], ScNets [HAL90], and ASOCS [MAR86]. ScNets aim at providing an alternative to knowledge elicitation from experts. Known rules may be pre-encoded and new rules can be learned inductively from examples. The representation lends itself to rule generation, but the constructed networks are complex and generalization does not appear trivial. In both PDLA and ASOCS, prior knowledge must also be explicitly encoded. FLARE's approach is more flexible. Because the system can reason, some prior knowledge can be deduced automatically from commonsense knowledge. Moreover, ASOCS and PDLA are limited to nominal domains. FLARE, like ScNets, handles both nominal and linear (including continuous) domains. Prior knowledge may take a variety of forms, some of which are discussed in [BUN90]. FLARE uses either pre-encoded defaults or deducible domain knowledge. FLARE's prior knowledge is often used as a learning bias to prune and constrain the search in the input space. A formal discussion on the need for such biases is in [MIT80].

FLARE's reasoning is a combination of default (i.e., rule-based) and similarity-based reasonings, as in CONSYDERR [SUN92]. However, CONSYDERR is strictly concerned with a connectionist approach to concept representation and commonsense reasoning. The resulting model is elegant. It consists of a two-level architecture that naturally captures the dichotomy between conceptual and subconceptual (i.e., features) levels. However, it does not address the problem of learning (how such a skill could be incorporated is also unclear), and is currently only instantiated at the conceptual level. FLARE's representation is not as elegant, but the model can effectively be instantiated both at the conceptual and subconceptual levels. CONSYDERR deals only with Boolean features and a concept's representation is limited to a single conjunction of features. FLARE's concepts are disjunctive conjunctions and the domain of features is not restricted. Hence, FLARE uses a more general distance metric than CONSYDERR's similarity measure based on feature overlap. However, FLARE has no mechanisms for individual weighting of features.

FLARE can be used as a simple expert system's inference engine. As an expert system, it can perform forward chaining in an attempt to reach a target goal. Unlike expert systems, whose knowledge base is encoded *a priori*, FLARE learns (and evolves) its knowledge base from both examples and rules. Hence, generalization may take place. There are currently no mechanisms in FLARE to deal explicitly with fuzzy rules. However, several mechanisms exist to handle inconsistencies and conflicts. FLARE always makes a decision based on available evidence. A confidence level can also be produced to characterize the "goodness" of the decision.

FLARE offers solutions to many useful commonsense protocols. Many solutions have been proposed to the inheritance problem for example (see [HOR90, SUN93, YOU93]). FLARE's solution is similar to CONSYDERR's [SUN93]. However, FLARE is able to handle its protocols either extensionally or intensionally. Most reasoning systems, because they do not have any inductive mechanisms, are confined to the intensional approach. The extensional approach endows the system with more self-adaptivity.

Work on Cyc [LEN90, LEN94] is also relevant to the context of the present research as it seeks to capture most of human commonsense knowledge into a large database that can be used by a variety of models such as FLARE. Relevant contexts of Cyc's knowledge base might be instantiated to guide FLARE's learning.

3. FLARE - A Framework for Learning and Reasoning

In this section, FLARE's learning and reasoning mechanisms are detailed, preceded by a description of how first-order clauses are translated into their equivalent attribute-value language representation and a discussion of the semantical difference between learned "classification" rules and "conceptual" implications.

3.1. First-Order to Attribute-Value Translation

Reasoning tasks are often expressed in English and have simple, direct counterparts in the classical first-order logic language (FOL). Because FLARE's representation language is the attribute-value language (AVL), FOL clauses must first be translated into their AVL equivalent. Note that AVL is not as expressive as FOL, so that FLARE has some inherent limitations. In particular, there is no support for variable binding and unification. Essentially, only FOL clauses of the form $(\forall x)p(x)\Rightarrow q(x)$, i.e., with a single (universally quantified) variable, may be translated. In some cases, this restriction may be partially lifted by contextual information and/or the addition of predicates (see Examples 2 and 4). Despite these limitations, FLARE effectively handles a wide range of applications. Moreover, AVL accounts for simple, efficient matching mechanisms, and lends itself naturally to many inductive learning problems. Indeed, many successful learning systems use AVL [CLP89, MIC83a, QUI86].

FOL statements of the aforementioned form are translated in a straightforward way into an equivalent symbolic-valued AVL representation, as follows.

1. Each predicate (e.g., on_table(x)) becomes a Boolean attribute.
2. Each "class" of objects (e.g., blocks) becomes a multi-valued (i.e., nominal) attribute, such that each of the attribute's values represents an instance of the class (e.g., A, B, and C if A, B, and C are blocks).
3. Each implication becomes a *vector* (defined over the cross-product of the domains of the attributes defined in steps 1 and 2) in which the attributes corresponding to the premise and the conclusion are asserted and all other attributes are don't-care (i.e., $*$). The attribute representing the conclusion or target is tagged as such (e.g., subscript T).

Note that steps 1 and 2 must be done first (i.e., the set of clauses must be "scanned" once) so as to identify all attributes. Dynamic binding of new attributes (for example in an incremental setting) is a topic of further research.

It is clear that as the number of predicates increases, so does the size of the vectors. Since all vectors are of the same size, and many of them may only have values set for a relatively small number of their attributes, this may result in large memory requirements, as well as in an increase of execution time of operations on vectors. A simple extension to the above transformation can be made to limit the growth of vectors. Predicates that qualify different possible values of the same concept (e.g., red(x), yellow(x), for colors) do not have to give rise to distinct Boolean attributes. They are instead combined into a single multi-valued attribute (e.g., color(x, V), where V is a constant: red, yellow, etc., see also Example 4). This is particularly useful for the conclusion part $q(x)$ as it may correspond to a classification for x.

Following are four simple examples that demonstrate the transformation.

Example 1: Nixon Diamond [REI81]

1. Republicans are normally not Pacifist	Republican(x)$\Rightarrow\neg$Pacifist(x)
2. Quakers are normally Pacifist	Quaker(x)$\Rightarrow$Pacifist(x)
3. Nixon is a Republican and a Quaker	Republican(Nixon), Quaker(Nixon)

Scan: 3 Boolean attributes for predicates Republican, Quaker and Pacifist

AVL Translation:

1. 1 * 0_T
2. * 1 1_T
3. 1 1 $?_T$ where ? stands for don't-know

Example 2: Root Node Definition [KUN94]

1. A root-node *is* some other node's parent but is not itself a child-node
   Parent(x,y),¬Child-node(x)⇒Root-node(x)
2. A child-node *has* some other node as a parent
   Parent(y,x)⇒Child-node(x)

Scan: 4 Boolean attributes for predicates Child-node, Root-node, *Has*-parent and *Is*-parent

AVL Translation:

1. 0 1_T * 1
2. 1_T * 1 *

Note that it is information about the semantics of the predicate Parent that makes things work in AVL. In particular, Parent gives rise to two different concepts depending on the order of its variables.

Example 3: Flying or Not Flying [LIF88]

1. Animals typically do not fly Animal(x)⇒¬Fly(x)
2. Birds are animals Bird(x)⇒Animal(x)
3. Birds typically fly Bird(x)⇒Fly(x)
4. Penguins are birds Penguin(x)⇒Bird(x)
5. Penguins do not fly Penguin(x)⇒¬Fly(x)

Scan: 4 Boolean attributes for predicates Animal, Bird, Penguin and Fly

AVL Translation:

1. 1 * * 0_T
2. 1_T 1 * *
3. * 1 * 1_T
4. * 1_T 1 *
5. * * 1 0_T

Example 4: Fitting Lenses

1. Low tear production rate causes dry eyes Tpr(x, low)⇒Eyes(x, dry)
2. Dry eyes should not have things put in them Eyes(x, dry),Thing(y)⇒¬Fit(x,y)
3. Lenses are things Lenses(y)⇒Thing(y)

Scan: 2 Multi-valued attributes for predicates Tpr and Eyes
3 Boolean attributes for Thing, Lenses and Fit

AVL Translation:

1. low dry_T * * *
2. * dry 1 * 0_T
3. * * 1_T 1 *

Note that it is contextual information about the relation between x and y that makes things work in AVL

To reason in the AVL context, special matching mechanisms are developed, and an execution scheme is devised to properly handle conflicting defaults and inheritance. FLARE's execution implements a non-monotonic default logic.

3.2. A Question of Semantics

As discussed above, FLARE only handles statements of the form $(\forall x)p(x)\Rightarrow q(x)$, where p is a conjunction of predicates on x. The semantics attached to such statements may be of at least two forms: *definition* and *implication*. If the predicates in p correspond to features of x, and q corresponds to some classification, or definition for x, then the statement is a definition. Rules obtained by induction from examples (i.e., learned classifications) represent such definitions. In the context of a definition, attributes (or predicates) of p may be viewed as subconceptual while q is conceptual. On the other hand, rules that relate concepts (at the same relative cognitive level) are implications. Such rules typically capture some domain knowledge, or may be generated by reasoning about the current knowledge base.

Definitions may be viewed as "if and only if" statements, while implications cannot. Induction, which produces definitions, lends itself naturally to the completion principle proposed in [CLK78]. That is, given a definition, if the knowledge base is instantiated at the conceptual level (i.e., q is known), then it is appropriate to assert p. However, the same does not hold for implications. For example, given that birds are animal, and that some x is an animal, it does not follow that x is a bird. The usefulness of the completion principle for definitions is in similarity-based reasoning, and the potential generation of new conceptual rules. Suppose, we have learned that $(\forall x)p(x) \Rightarrow q(x)$ and $(\forall x)p'(x) \Rightarrow q'(x)$, where p and p' overlap (i.e., have some predicates in common). If we now assert that $q(x)$, then, using the completion principle and subsequently asserting p(x), we may be able to derive $q'(x)$ based on the amount of overlap between p and p'. A new "conceptual" implication may thus be generated.

Consider an example from [COL89]. Assume it is known that west Texas is cattle country, and the question is whether the Chaco is also cattle country. If it is known that the Chaco is similar to west Texas (in some relevant way), then it may be plausibly concluded that the Chaco is cattle country (even though this conclusion is less certain than the one about west Texas). Using the above notation, we have $p(x) \Rightarrow \text{theChaco}(x)$, $p'(x) \Rightarrow \text{westTexas}(x)$, and $\text{westTexas}(x) \Rightarrow \text{CattleCountry}(x)$. Asserting theChaco (at the conceptual level) causes its definition p (at the subconceptual level) to be asserted, and then the overlap (which captures the similarity) between p and p' allows westTexas to be derived first, and finally CattleCountry.

In FLARE, the representation is extended and a *definition indicator* is tagged to those statements that may be completed (i.e., definitions, inductively learned classifications). Note that the need for such a (somewhat cumbersome) extension stems from the fact that FLARE does not physically separate conceptual and subconceptual levels. CONSYDERR on the other hand provides natural support for the dichotomy. FLARE's representation makes learning more readily applicable and preserves consistency with previously developed models. At this point, the issue of achieving both the dichotomy and easy learning remains open.

3.3. FLARE's Reasoning

FLARE implements a form of default reasoning combined with similarity-based reasoning, similar to CONSYDERR [SUN92]. Such a combination decreases the system's susceptibility to brittleness. It has also been shown to effectively account for several useful commonsense protocols, such as inheritance (see [SUN92], Appendix A).

Because FLARE uses AVL, its knowledge base is a set of vectors, and similarity is computed as a distance function in the corresponding vector space. To initiate reasoning, a new vector is created in which attributes whose values are known are asserted, while all other attributes are set to ? (i.e., don't-know). The system then uses its knowledge base to derive as many conclusions as possible based on given information.

Let v be the vector representing the current state of knowledge of the system. One attribute, designated as T-attribute, is chosen as the actual, final target (sometimes referred to as top goal) for the current context. To derive a conclusion (i.e., set the T-attribute of v to an appropriate value), two mechanisms are available: rule application and similarity-based assertion. If a rule exists that can be applied, then it is applied. It is only when no rule can be applied that similarity-based reasoning takes effect. The following sections describe these mechanisms, and detail FLARE's reasoning scheme.

To illustrate the definitions and algorithms of sections 3.3 and 3.4, we construct an example based on the *mediadv* knowledge base [HAR85]. This purposely simple example will serve as a running example throughout the paper. A discussion of the complete *mediadv* knowledge base is in Section 5.3. Here, two conditions (i.e., instructional feedback and presentation modification) have been left out, and many of the disjunctions predicting the stimulus situation and the appropriate response have been omitted. The goal is for the system to predict the media to consider, based on 4 conditions: stimulus situation, appropriate response, stimulus duration, and training budget. Only a few of the original rules are used, and some are more general to illustrate the use of examples and precepts (see Section 3.4.1). A few examples have also been added to exercise important features of the algorithms. Table 1 shows the informal English version of the knowledge (with reference to the rule of *mediadv* it was generated from when applicable) used and its corresponding translation into AVL vectors. We let *KB* be the resulting set of vectors. The attributes are given in the order: situation, stimulus-situation, response, appropriate-response, stimulus-duration, training-budget, and media-to-consider. Note that all the attributes are nominal. The symbolic values used in the English statements are transformed into equivalent nominal values in the vectors. Hence, for example, the first statement gives rise to a vector in which the attribute situation is set to 0 (the corresponding nominal value of schematics for this attribute), and the target attribute stimulus-situation is set to 0 (the corresponding nominal value of symbolic for this attribute).

Table 1 - Simple *KB* Running Example

English Statements	Equivalent AVL Vectors
If situation = schematics then stimulus-situation = symbolic (R3)	v_1 = 0 0_T * * * * *
If situation = conversation then stimulation-response = verbal (R4)	v_2 = 1 1_T * * * * *
If situation = photograph then stimulus-situation = pictorial (R2)	v_3 = 2 2_T * * * * *
If response = observing or response = thinking then appropriate-response = covert (R5)	v_4 = * * 0 0_T * * * v_5 = * * 1 0_T * * *

If response = emoting then appropriate-response = affective (R10)	v_6 = * * 2 1_T * * *
If stimulus-situation = verbal and appropriate-response = covert and stimulus-duration = brief then media-to-consider = lecture (R13)	v_7 = * 1 * 0 0 * 2_T
If stimulus-situation = verbal or stimulus-situation = symbolic or stimulus-situation = pictorial and appropriate-response = covert and stimulus duration = brief and training-budget = medium then media-to-consider = lecture-with-slides (R14)	v_8 = * 1 * 0 0 1 3_T v_9 = * 0 * 0 0 1 3_T v_{10} = * 2 * 0 0 1 3_T
If stimulus-situation = verbal and stimulus-duration = brief then media-to-consider = role-play-w/verbal-feedback (R16)	v_{11} = * 1 * * 0 * 0_T pty = 1
If stimulus-situation = verbal and appropriate-response = affective then media-to-consider = role-play-w/video-feedback (R17)	v_{12} = * 1 * 1 * * 1_T pty = 3
If situation = conversation and response = observing or response = thinking and stimulus-duration = brief and training budget = medium then media-to-consider = lecture	v_{13} = 1 * 0 * 0 1 2_T v_{14} = 1 * 1 * 0 1 2_T
If situation = photograph and response = emoting and stimulus-duration = persistent and training-budget = small then media-to-consider = role-play-w/verbal-feedback	v_{15} = 2 * 2 * 1 0 0_T
If situation = photograph and response = emoting and stimulus-duration = persistent and training-budget = small then media-to-consider = lecture	v_{16} = 2 * 2 * 1 0 2_T
If situation = photograph and response = emoting and stimulus-duration = persistent and training-budget = small then media-to-consider = role-play-w/verbal-feedback	v_{17} = 2 * 2 * 1 0 0_T

<u>3.3.1. Rule Application</u>. A rule may be applied if it *covers v*. A vector x is said to cover a vector y if and only if x and y have the same T-attribute, and the set of remaining attribute-value pairs of x is a subset of the set of remaining attribute-value pairs of y. For example, in *KB*, v_{11} covers v_7 and v_8, but v_{11} does not cover v_9 or v_{12}. Intuitively, x covers y if y satisfies all of the premises of x.

The notion of subset is extended slightly to accommodate real-valued attributes. Given that the probability of two real values being equal is extremely small, the cover relation would essentially never hold. The notion of equality is extended as follows. Two values x_1 and x_2 are *equal* if and only if $|x_1-x_2| \leq$, for some > 0 (see [GIR94d] for details). Hence, the vector (*, 1.2, 3.52, *, 0_T) covers the vector (2, 1.3, 3.48, *, 0_T) if = .5. In the current implementation is some fraction of the range of possible values of each attribute.

3.3.2. Similarity-Based Assertion. The following non-symmetric distance function is defined over vectors. Vector $\boldsymbol{x}$ is stored in the knowledge base, and vector $\boldsymbol{y}$ is presented to the system to reason about. The distance is always computed from $\boldsymbol{x}$ to $\boldsymbol{y}$.

$$D(\boldsymbol{x},\boldsymbol{y}) = \frac{\displaystyle\sum_{i=1}^{n} \begin{cases} x_i \neq y_i & \text{if } x_i, y_i \neq *, ? \text{ and attribute } i \text{ is nominal} \\ \dfrac{|x_i - y_i|}{range(i)} & \text{if } x_i, y_i \neq *, ? \text{ and attribute } i \text{ is linear} \\ .5 & \text{if } (x_i = ?) \vee [(x_i \neq *) \wedge ((y_i = *) \vee (y_i = ?))] \\ 0 & \text{otherwise} \end{cases}}{num\_asserted(\boldsymbol{x})}$$

where $range(i)$ is the range of values of attribute i, and $num\_asserted(\boldsymbol{x})$ is the number of attributes that are not $*$ in $\boldsymbol{x}$

A detailed discussion of and partial justification for the definition of D is in [GIR94c]. $D(x,y)$ is meaningful only if x and y have the same T-attribute, and the T-attribute is left out of the computation. For example, $D(v_{11}, v_8) = 0$, $D(v_{13}, v_{14}) = 1/5$, $D(v_7, v_{16}) = 2/3$, $D(v_{16}, v_7) = 5/8$, and $D(v_1, v_4)$ is undefined.

Similarity-based assertion consists of asserting the T-attribute of v to the value of that attribute in v's closest match. D extends the similarity function defined in [SUN92] to non Boolean attributes. It relies on distance between values rather on the amount of overlap between Boolean attributes. If all attributes are Boolean and there are no missing values, then D is actually the reciprocal of the aforementioned similarity function.

Note that (since D is not symmetric) x covers y if and only if $D(x,y) = 0$. Hence, since 0 is the minimum of the distance function, D can be used to apply both reasoning mechanisms with the correct order (i.e., rules first, similarity next), by computing the distance from all the rules in the current knowledge base to v, and simply selecting the rule that minimizes D. As it is possible that more than one rule minimizes D, a priority scheme is devised to choose a winner. This conflict resolution procedure is outlined in Section 3.3.3.

3.3.3. Conflict Resolution. When more than one rule minimizes D (i.e., can be selected for application), a winner is chosen according to the following priority scheme, where each subsequent condition is invoked if a tie exists at the previous level.

1. most specific
2. highest static priority
3. highest dynamic priority
4. greatest cover

Specificity is defined to be the number of attributes, other than the T-attribute, whose value is not $*$. A vector x is more specific than a vector y if specificity(x) > specificity(y). For example, in *KB*, v_7 has specificity 3, v_8 has specificity 4, and v_{13} has specificity 5, so that v_{13} is more specific than both v_7 and v_8. The static priority value may be set as rules are learned (e.g., v_{11} and v_{12} in *KB*). It allows the intensional handling of conflicts.

The dynamic priority is updated through learning. It allows the extensional handling of conflicts. The cover of a vector x is the number of other vectors in the current knowledge base that x covers and that have the same T-attribute's value. It can be viewed intuitively as a confidence level in the validity of the rule, based on available evidence. In *KB*, all vectors have cover 0, except v_7 and v_{11} that have covers 1 and 2, respectively.

Giving priority to the most specific vector allows FLARE to handle exceptions and cancellation of inheritance. The static priority scheme makes it possible to handle conflicting defaults as defined by a teacher. The dynamic priority scheme accounts for epistemological inconsistencies that may be resolved over time as more information becomes available in support of one belief or the other (see Section 3.4.2). Finally, selecting the vector with greatest cover allows evidence gathered from experience to guide a final selection. Note that the current scheme gives precedence to teacher-provided information. Other priority schemes can easily be defined. For example, static priorities could be given as a simple form of initial bias, and evidence gathered through learning (e.g., dynamic priority and cover) could be used to confirm or modify these priorities.

3.3.4. Completion. The above description and the use of D assumes that the model is instantiated at the subconceptual level. As discussed in Section 3.2., it is sometimes useful to instantiate the model at the conceptual level. Each vector is tagged with a flag that characterizes the rule it represents as either an implication or a definition. Before FLARE performs any of its other reasoning mechanisms, it finds all asserted attributes of v that are T-attributes of definitions in the current knowledge base. If any such attribute is found, and for all of them, completion is applied. That is, all asserted attributes of the corresponding definitions that are missing (i.e., ?) in v, are "copied" into v. Potential conflicts are resolved by giving precedence to the first copy made. The algorithm is as follows.

Completion

1. For each non-asserted attribute a of v other than the T-attribute
   If a is the T-attribute of some definition d and their values are equal
   Then, copy all asserted attributes of d that are still ? in v, into v

Because FLARE's concepts and rules are disjunctive conjunctions, where each vector is a conjunction and all the vectors having the same T-attribute form the disjunction, a choice must be made for the application of the completion principle. Note that each conjunction need not be mutually exclusive with other conjunctions. Currently, the system chooses one conjunction at random to perform the completion on. Other mechanisms (e.g., apply to all, select a winner based on some criteria, etc.) are the topic of further research.

3.3.5. Forward Chaining. As mentioned above, v has a single T-attribute, corresponding to the final goal to achieve. However, at any given time, any one of the (yet) non-asserted attributes of v may be designated as a temporary T-attribute. A temporary T-attribute is called a t-attribute. A t-attribute represents a possible subgoal that may be useful (or necessary) in reaching the final goal. FLARE then performs forward chaining as follows.

Forward_chaining

1. Do

a. Let $w = v$
{ create a temporary copy of v }
b. For each non-asserted attribute[3] a of v other than the T-attribute
i. Designate a as a t-attribute
ii. If a rule can be applied to $\boldsymbol{v}$ to assert a, apply it by asserting a in $\boldsymbol{w}$
{ based on v, assert all possible attributes in w }
c. Let $v = w$
{ copy result back into v for the next level of inference }
Until no new attribute of v has been asserted
2. If a rule can be applied to assert v's T-attribute, then apply it
Else perform similarity-based assertion

Step 1 is the heart of the algorithm. Each execution of step b corresponds to the achievement of all possible subgoals at a given depth in the inference process. Each iteration uses knowledge acquired in the previous iteration to attempt to derive more new conclusions. Consider *KB*, and assume the vector $v = (1, ?, 1, ?, 0, 0, ?_T)$ is input to the above algorithm. Execution proceeds as follows. There are only 2 steps in the forward chain.

Step 1. a. Let $w = v$
b. i. Designate the second attribute as a t-attribute (first subgoal)
b. ii. Apply rule v_2: result is $w = (1, 1, 1, ?, 0, 0, ?_T)$
b. i. Designate the fourth attribute as a t-attribute (second subgoal)
b. ii. Apply v_5: result is $w = (1, 1, 1, 0, 0, 0, ?_T)$
c. Let $v = w$
Step 2. Compute distance to all vectors in *KB*
Two conflicting winners: $D(v_7, v) = D(v_{11}, v) = 0$
Apply conflict resolution: v_7 is more specific so it wins
Apply v_7: result is $v = (1, 1, 1, 0, 0, 0, 2_T)$

Notice that only rules are applied for the assertion of t-attributes (i.e., subgoals). As a result, the accuracy of the final goal is increased, but the ability to perform approximate reasoning is reduced. It is possible to relax this restriction thus potentially achieving more subgoals, but reducing the confidence in the final result. For example, step ii could be modified to allow not only rules, which are perfect matches (i.e., $D = 0$), but also matches deemed to be "close enough." The measure of closeness can be implemented via a threshold value T_D, placed on D. That is, step ii is replaced with:

ii_1'. Find closest match (according to Section 3.3.3) - distance D'
ii_2'. If $D' \leq T_D$ perform similarity-based assertion (rule application if $D' = 0$)

---

[3]These are either $*$ or ?. They are $*$ when precepts and rules with differing premises and conclusions are used. In such cases, it is not clear until reasoning whether they are true don't-cares (see Section 3.4.1) or only don't-knows.

The value of T_D then offers a simple mechanism to increase the level of approximate reasoning. This is particularly useful for cases such as the Chaco example, where, after completion, some of the reasoning is based on the amount of similarity between concepts. Notice that steps ii_1' and ii_2' are functionally equivalent to step ii when $T_D = 0$.

Step 2 concludes the reasoning phase by asserting the target attribute (or goal). The goal is always asserted, either by rule application or similarity-based assertion. Hence, FLARE always makes a decision. The accuracy of the decision depends upon available information[4].

3.3.6. FLARE's Reasoning Algorithm. Given all of the above mechanisms, FLARE reasons as follows. Let *v* be the vector containing the facts to reason about.

Reasoning

1. Perform Completion
2. If *v*'s T-attribute has been asserted, then done
   Else Perform Forward_chaining

Step 2 causes reasoning to stop if the goal can be achieved by completion based on existing definitions. This is consistent with the semantics of definitions, and allows the system to reach subconceptual goals that are not achievable by existing rules.

In order to test the predictive accuracy of FLARE, the standard training set/test set approach may be used. The value of *v*'s T-attribute is provided, but it is not used during reasoning. Rather, the system reasons based on its current knowledge base and all of the asserted attributes of *v*. When reasoning is completed, the "computed" target value is compared with the "actual" target value. More information may also be displayed to show how the goal is achieved. Currently, FLARE is non-interactive, that is, it cannot query a user for the values of missing attributes that may help improve the accuracy of its result.

3.4. FLARE's Learning

The kind of learning supported by FLARE is supervised. Thus, a target output is provided along with each training item. Learning is incremental. Training items become available one at a time, over time. The system continually adapts to the new information it receives. As is inherent in nature, some information may be noisy, and certain pieces of information may be encountered more than once. Moreover, FLARE extends inductive learning from examples with prior knowledge in the form of precepts. Section 3.4.1 contains some preliminary definitions. Section 3.4.2 presents FLARE's learning algorithm. Section 3.4.3 discusses how generalization is effected. Section 3.4.4 shows the details of the adaptation phase of the learning algorithm The *KB* example of Section 3.3 is

---

[4]The accuracy, or confidence level may be computed in a variety of ways from information about static priorities, dynamic priorities, covers, and counters (see Section 3.4). It may be given relative to other alternatives (e.g., this decision is twice as reliable as this other one based on their relative static or dynamic priorities), or in absolute (e.g., this decision has a confidence of 95% based on counters). A wide range of possibilities is available.

used throughout, and Section 3.4.5 details how FLARE would learn *KB*. Finally, Section 3.4.6 addresses the issue of extensionality and intensionality.

3.4.1. Examples vs. Precepts vs. Rules. Induction consists of going from the specific to the general. The setting typically involves some target concept and a set of attributes whose values are believed to be relevant in predicting the values of the concept. In this context, the target concept is a T-attribute. The relevant attributes shall be called R-attributes. Note that each T-attribute thus has a unique associated set of R-attributes. For example, the vectors of *KB* can be used to predict the value of the attribute media-to-consider. We shall designate that attribute as the T-attribute, and attributes stimulus-situation, appropriate-response, stimulus-duration, and training-budget as the R-attributes in this context.

An *example* of a concept *C* is a vector that has all of *C*'s R-attributes set to either ? or one of their possible values. The special symbol ? accounts for possible missing attribute values in real-world examples. *Precepts* and *rules* are vectors in which some of the R-attributes are set to the special value *don't-care* (∗). In *KB*, v_8, v_9, and v_{10} are examples, while v_7 and v_{12} are precepts.

The don't-care value has the following semantics. If i_j is an R-attribute whose value is don't-care in a vector p, then every vector p' obtained from p by asserting i_j to any one of its possible values and leaving p's other R-attributes unchanged has the same output (or concept) value as p. Informally, this says that in the context of p, the value of i_j has no effect on the value of the output. A vector containing don't-care R-attributes *represents* several examples. Consider the following example.

> Let there be 3 R-attributes ranging over the set {0, 1, 2}
> Let the T-attribute be Boolean
> Then the vector $(*, 1, 0, 0_T)$ represents (or covers) all of the following:
> $(0, 1, 0, 0_T)$ $(1, 1, 0, 0_T)$ $(2, 1, 0, 0_T)$

The distinction between rules and precepts is limited to learning. In reasoning, all vectors (including examples that do not generalize) are rules. In learning, the term rule is applied to an induced generalization from examples. In FLARE, rules are formed by applying the dropping condition generalization rule [MIC83a], that is, under certain conditions (see Section 3.4.3), one attribute is set to don't-care. The term precept is applied to a generalization given either by a teacher or deduced from an instantiation of commonsense knowledge to the domain under study. Precepts reflect some high-level knowledge (or common sense) about the real-world. A precept "suggests something advisory and not obligatory communicated typically through teaching" (Webster's Dictionary). Precepts are not necessarily correct, but convey some hint or bias. For instance, when dealing with how weather conditions affect what one wears, a possible precept is: "if it rains, take an umbrella," regardless of the temperature, the day of the week, or any other unnecessary (don't-care) detail.

Seminal work on the need for biases [MIT80], the strong knowledge principle [WAT86], and empirical results on the use of precepts to augment inductive learning

[GIR93b] suggest the validity of using both raw examples and prior knowledge to improve the performance of learning systems.

3.4.2. FLARE's Learning Algorithm. The set of all examples, rules and precepts that share the same R-attributes and T-attribute can be viewed as a function mapping the R-space into the T-space. In this context, an example maps a single point of the R-space to a value in the T-space, while precepts and rules are hyperplanes of the R-space that map all of their points to the same value in the T-space.

Learning then follows a form of nearest-hyperplane learning. It consists of first applying the reasoning scheme, and then making adjustments to the current knowledge base to reflect the newly acquired information. Note that forward reasoning on partial information is effected prior to the induction. Hence, if there are missing attributes and the knowledge base contains rules that can be applied to assert these attributes, the rules are applied so that as many of the missing attributes as possible are asserted before the final target is predicted. Hence, the accuracy of the prediction is increased. Learning proceeds as follows.

Learning

1. Perform Reasoning[5]
2. Perform Adapting

The reason the algorithm is said to be nearest-hyperplane is that the reasoning phase essentially identifies a closest match for the (possibly filled-in) incoming vector. This closest match is either a rule (i.e., true hyperplane) or a stored example (i.e., point or degenerated hyperplane). The scheme for choosing a closest match is discussed in Section 3.3.

The system starts with an empty knowledge base. It then adapts to each new vector *v*, where *v* is either a precept or an example. If *v* is the first vector, then there is no closest match and *v* is automatically stored in the current knowledge base. In that sense, the first vector represents yet another bias for the learning system. If *v* is not the first vector, then a closest match, say *m*, is found and the knowledge base adapts itself (step 2) based on the relationship between *v* and *m*. Note that *m* is closest given the current available information, and can indeed be "far" from *v*. Hence, the order in which training takes place impacts the outcome. Chronology could therefore be used as another learning bias. An example of the implicit use of chronology to always give priority to new information is in PASOCS [MAR94].

Notice that the adaptation phase takes place regardless of the target value predicted by the reasoning phase. A possible alternative would be to adapt only if the predicted target value differs from the actual target value. It has been found empirically, however, that too much useful information is lost with this approach due to the incrementality of the system and its sensitivity to ordering. A possibly viable alternative would make use

[5]Reasoning based upon available knowledge prior to adapting is plausible. Even when available information is insufficient and/or incomplete, humans often attempt to make a tentative decision and get corrected if necessary. The more information becomes available, the more accurate decisions become. But, even in a don't-know state, humans (if they have to) are able to make a best guess that corresponds to their current representation of the world.

of memory. Vectors currently accounted for could be saved in memory and presented later to the system. This may be done a few times over some period of learning time until either the vectors must be stored in the knowledge base (due to changes in the knowledge base) or they are discarded as the system has gained enough confidence in its ability to account for them. The problem being addressed here is the inherent non-monotonicity of learning and reasoning.

3.4.3. Generalization. Two vectors can produce a generalization when the following five conditions hold.

1. they differ in the value of exactly one of their R-attributes
2. the R-attribute on which they differ is nominal
3. they are concordant
4. the number of their R-attributes not equal to $*$ differ by at most 1[6]
5. at least one of them has more than one non $*$ R-attribute

Generalization then consists of setting to $*$ the R-attribute on which the two vectors differ, in the vector that is most general, as long as that vector has more than one non $*$ R-attribute. For example, vectors v_8 and v_9 of *KB* satisfy the above conditions and would generalize to produce a vector $v_{8\text{-}9} = (*, *, *, 0, 0, 1, 3_T)$.

Choosing the most general vector maximizes generalization, and the condition on the number of non $*$ R-attributes guarantees that no rule is generated that would cover every other vector. This rule is only applied to nominal attributes as it makes little sense for linear (especially real-valued) domains. For linear attributes, generalization is achieved through an artifact due to the extension of the notion of equality shown in Section 3.3.1 (see Note 5 in Section 3.4.4).

Though dropping the condition may be argued to also be too strong for large nominal domains, it is balanced by the system's ability to handle exceptions. Notice however that given a set of vectors of the form $v_i = S\, x_i\, T_k$ where S is fixed, T is the T-attribute, and $x_i \neq x_j$ for all $i \neq j$, any pair of concordant vectors satisfy the generalization condition, yet only the first such pair will generalize. All other vectors then become either covered by this generalization or exceptions to it. If most are exceptions, this leads to the storage of more vectors than needed, especially for large domains where various subsets of values give rise to different T-attribute's values. Moreover the outcome depends upon ordering of the vectors.

Also, if there exists conflicts involving one (or more) value of x, then the system will end up giving unfounded precedence to the exceptions (being more specific), and again these depend on the ordering. Support for internal disjunction, or a more complex generalization scheme may help alleviate some of these problems. They are the topic of future research. The current simple generalization scheme gives promising empirical results for induction (see Section 4).

3.4.4. Adapting. Along with each vector, FLARE stores the following information.

---

[6]This value is based upon empirical evidence.

- a static priority value (*pty*),
- a dynamic priority value (*dyn_pty*),
- the number of vectors covered (*num_covers*),
- a definition indicator, and
- a set of counters (*counters*[]) for each possible value of the T-attribute.

The value of pty is set to 0 by default, but may be set to any other value *a priori*. The value of num-covers records the number of training items covered by the vector, and is a kind of confidence level for that vector. The definition indicator is a Boolean flag that identifies the vector as being a definition. It is used to perform completion during reasoning. The counters are all initialized to 0, except the one corresponding to the vector's T-attribute value which is initialized to 1. The counters evolve over time, and are used to handle noise. For any vector *p* in the current knowledge base, exactly one counter value is incremented (by 1) each time a new vector is presented, whose R-attributes' values are all equal to those of *p*. The value incremented corresponds to the new vector's T-attribute value. The counter value that is highest represents the statistically "most probable" T-attribute value. In effect, the T-attribute value of a vector is always the one with highest count. Note that this value may change over time, as new information becomes available.

The value of dyn_pty is initialized to 0. As it is intended to resolve conflicting defaults extensionally, its evolution requires the identification of such conflicts. Identification takes place in the reasoning phase as FLARE discovers two rules (or precepts) that apply equally well to the new vector. Two rules *R* and *S* are in conflict for a vector *v* if the following conditions hold.

1. $D(R,v) = D(S,v) = 0$
2. *R* and *S* have the same specificity
3. *R* and *S* have the same static priority
4. *R* and *S* have different T-attribute's value
5. *R* and *S* overlap (i.e., the set of all possible vectors they cover intersect)

When reasoning and coming upon conflicting defaults, FLARE increments by 1 the value of dyn_pty of that default whose T-attribute's value is the same as *v*'s (i.e., the default that covers *v*). If none of the defaults cover *v*, then no change is made. Reasoning then proceeds as outlined in Section 3.3.3. Notice that dyn_pty evolves over time so that the system's response also changes based on accumulated evidence. Note that if none of the defaults cover *v*, then *v* is an exception to these defaults, and thus it will be stored in the knowledge base (see algorithm Adapting below). In turn, the resolution of conflicts would be handled by the counters rather than dyn_pty due to the reasoning's priority scheme. Functionally, the result is identical.

The details of step 2 (i.e., Perform Adapting) of the learning algorithm are as follows. Let *v* be a new vector, other than the first one, and let *m* be *v*'s closest match in the current knowledge base. The vectors *v* and *m* are said to be *concordant* if they have the same T-attribute's value.

Adapting

```
If (v.R-attributes' values = m.R-attributes' values)[7]        { duplicates or noise }
    add 1 to m.counters[v.T-attribute's value]
Else if (m covers v and m and v are concordant)
    add 1 to m.num_covers
Else if (v covers m and m and v are concordant)
    add 1 to v.num_covers
    delete m from the knowledge base
    add v to the knowledge base
Else if (v and m can produce a generalization)
    If (v is more specific than m and m has more than one non ∗ R-attribute)
        drop the condition in m
        m.pty = max(m.pty, v.pty)
    Else if (v has more than one non ∗ R-attribute)
        drop the condition in v
        v.pty = max(m.pty, v.pty)
        v.num_covers = m.num_covers
        delete m from the knowledge base
        add v to the knowledge base
    Else add v to the knowledge base
Else add v to the knowledge base
```

Because a best match is first identified, changes to the knowledge base are localized, and are guided by the kinds of possible relationships between *v* and *m*. These relationships are summarized below.

- *v* is identical to *m* (i.e., noise or duplicates), or
- *v* is covered by *m* (i.e., *v* is a special case of *m*), or
- *v* covers *m* (i.e., *v* is a general case of *m*), or
- *v* and *m* can produce a generalization, or
- all other cases (e.g., *v* is an exception to *m*, *v* and *m* are too far apart, etc.)

In the first two cases, there is no need to store *v* as the current knowledge base has sufficient information to correctly predict *v*'s target value. In the third case, *v* is stored and *m* removed as *v* is more general than *m*, and thus accounts for it. In the fourth case, only one of *v* or *m* is generalized and stored. The values of pty and num_covers are set so that the generalization inherits the maximum static priority value and the current value of num_covers. Finally, in the fifth case, *v* must be added as the current knowledge base either does not produce the correct target value for *v* (e.g., exceptions), or is not deemed reliable enough to properly account for *v*.

---

[7]Note that the extension of the notion of equality discussed in Section 3.3.1 allows this part of the algorithm, in conjunction with the counters, to produce some generalization for linear attributes. In effect, the vector retained in the knowledge base acts as a "prototype," and its T-attribute's value is the one most probable among its neighbors.

<u>3.4.5. A Detailed Example: Learning *KB*</u>. This section shows how FLARE would learn *KB*. It highlights several interesting features of both reasoning and adaptation. Let *KB′* be the current knowledge base of FLARE. As discussed above, FLARE starts with *KB′* = ∅. Now assume that each vector in *KB* is presented to FLARE in the order in which it appears in *KB*. Also, assume that vectors v_7 through v_{17} are tagged as definitions.

1. *Presentation of* v_1. *KB′* = ∅. v_1 is simply added to *KB′*.

2. *Presentation of* v_2. *KB′* = $\{v_1\}$. v_1 is the closest match. v_1 and v_2 are not concordant, so v_2 is added to *KB′*.

3. *Presentation of* v_3. *KB′* = $\{v_1, v_2\}$. Same as above with either v_1 or v_2. So v_3 is added to KB'.

4. *Presentation of* v_4. *KB′* = $\{v_1, v_2, v_3\}$. No winner can be found since none of the vectors in *KB′* have the same T-attribute as v_4. So v_4 is added to *KB′*.

5. *Presentation of* v_5. *KB′* = $\{v_1, v_2, v_3, v_4\}$. v_4 is the only (and hence closest) match since none of the other vectors in *KB′* have the same T-attribute as v_5. v_4 and v_5 satisfy conditions 1-4 for generalization but violate condition 5, so v_5 is added to *KB′*.

6. *Presentation of* v_6. *KB′* = $\{v_1, v_2, v_3, v_4, v_5\}$. Same as step 3. So v_6 is added to *KB′*.

7. *Presentation of* v_7. *KB′* = $\{v_1, v_2, v_3, v_4, v_5, v_6\}$. Note that v_7 is a precept. No winner can be found since none of the vectors in *KB′* have the same T-attribute as v_7. So v_7 is added to *KB′*.

8. *Presentation of* v_8. *KB′* = $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$. v_7 is the only (and hence closest) match since none of the other vectors in *KB′* have the same T-attribute as v_8. v_8 is an exception to v_7 since v_7 covers v_8 but they are not concordant. Hence, v_8 is added to *KB′*.

9. *Presentation of* v_9. *KB′* = $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$. Only v_7 and v_8 may compete. $D(v_7, v_9) = 1/3$ and $D(v_8, v_9) = 1/4$. Hence, v_8 wins. v_8 and v_9 satisfy all five conditions for generalization. The second attribute is dropped (i.e., replaced by $*$) in either one, say v_8, to produce $v_8' = (*, *, *, 0, 0, 0, 3_T)$. v_8' is added to *KB′*.

10. *Presentation of* v_{10}. *KB′* = $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8'\}$. Only v_7 and v_8' may compete. $D(v_7, v_{10}) = 1/3$ and $D(v_8', v_{10}) = 0$. Hence, v_8' wins. v_8' covers v_{10} and they are concordant, so FLARE adds 1 to num_covers(v_8'). v_{10} need not be added to *KB′*.

11. *Presentation of* v_{11}. *KB′* = $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8'\}$. Notice that v_{11} is also a precept, so that precepts may be given at any time during learning. Only v_7 and

v_8' may compete. $D(v_7, v_{11}) = 1/6$ and $D(v_8', v_{11}) = 1/3$. Hence, v_7 wins. Neither one covers the other; they are not equal; they cannot produce generalization (violate condition 3). Thus, v_{11} is added to *KB'*. Note that v_{11} has a static priority of 1.

12. *Presentation of* v_{12}. $KB' = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8', v_{11}\}$. v_7, v_8', and v_{11} compete. $D(v_7, v_{12}) = 1/2$, $D(v_8', v_{12}) = 2/3$, and $D(v_{11}, v_{12}) = 1/4$. Hence, v_{11} wins. Neither one covers the other; they are not equal; they cannot produce generalization (violate condition 3). Thus, v_{12} is added to *KB'*. Note that v_{12} has a static priority of 3.

13. *Presentation of* v_{13}. $KB' = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8', v_{11}, v_{12}\}$. In this case, some non-asserted attributes of v_{13} may be asserted through reasoning, before the T-attribute. Rules v_2 and v_4 are applied to assert the second and fourth attributes respectively. The result is $v_{13}' = (1, 1, 0, 0, 0, 1, 2_T)$. v_7, v_8', v_{11}, and v_{12} compete to assert the T-attribute. $D(v_7, v_{13}') = 0$, $D(v_8', v_{13}') = 0$, $D(v_{11}, v_{13}') = 0$, and $D(v_{12}, v_{13}') = 1/2$. Both v_7 and v_8' win over v_{11} since they are more specific. However, v_7 and v_8' have the same specificity. In fact, they satisfy all five conditions that identify them as conflicting defaults in the current context. Hence, FLARE adds 1 to dyn_pty(v_7) since v_7 and v_{13}' are concordant. Reasoning then proceeds, giving precedence to v_7. Since v_7 covers v_{13}' and they are concordant, v_{13}' need not be added to *KB'*.

14. *Presentation of* v_{14}. $KB' = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8', v_{11}, v_{12}\}$. Some non-asserted attributes of v_{14} may be asserted through reasoning, before the T-attribute. Rules v_2 and v_5 are applied to assert the second and fourth attributes respectively. The result is $v_{14}' = (1, 1, 1, 0, 0, 1, 2_T)$. The rest is identical to step 13. Now, dyn_pty(v_7) = 2, and v_{14}' is not added to *KB'*.

15. *Presentation of* v_{15}. $KB' = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8', v_{11}, v_{12}\}$. Some non-asserted attributes of v_{15} may be asserted through reasoning, before the T-attribute. Rules v_3 and v_6 are applied to assert the second and fourth attributes respectively. The result is $v_{15}' = (2, 2, 2, 1, 1, 0, 0_T)$. v_7, v_8', v_{11}, and v_{12} compete to assert the T-attribute. $D(v_7, v_{15}') = 1$, $D(v_8', v_{15}') = 1$, $D(v_{11}, v_{15}') = 1$, and $D(v_{12}, v_{15}') = 1/2$. Hence, v_{12} wins. Neither one covers the other; they are not equal; they cannot produce generalization (violate condition 3). Thus, v_{15}' is added to *KB'*.

16. *Presentation of* v_{16} *and* v_{17}. $KB' = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8', v_{11}, v_{12}, v_{15}'\}$. Both are equal to v_{15}'. Neither v_{16} nor v_{17} need be added to KB', but the appropriate counter values are incremented in v_{15}'. The result is counters[0] = 2, counters[1] = 0, counters[2] = 1, and counters[3] = 0. Thus, the T-attribute's value of v_{15}' is currently 0.

The resulting *KB'*, after learning *KB* is as follows. The variables *p*, *c*, and *dp* stand for static priority, cover number, and dynamic priority, respectively.

v_1 = 0 0_T * * * * * (p=c=dp=0) v_7 = * 1 * 0 0
* 2_T (p=c=0, dp=2)

v_2 = 1 1_T * * * * *	(p=c=dp=0)	v_8' = * * * 0
0 1 3_T (p=dp=0, c=1)		
v_3 = 2 2_T * * * * *	(p=c=dp=0)	v_{11} = * 1 * *
0 * 0_T (p=1, c=dp=0)		
v_4 = * * 0 0_T * * *	(p=c=dp=0)	v_{12} = * 1 * 1
* * 1_T (p=3, c=dp=0)		
v_5 = * * 1 0_T * * *	(p=c=dp=0)	v_{15}' = 2 2 2 1
1 0 0_T (p=c=dp=0)		
v_6 = * * 2 1_T * * *	(p=c=dp=0)	

To further demonstrate reasoning, assume that a new vector $v = (1, ?, 0. ?, 0, 1, ?_T)$ is presented to FLARE after KB' is acquired. The second and fourth attributes are asserted as previously discussed to give $v' = (1, 1, 0, 0, 0, 1, ?_T)$. Then, v_7, v_8', and v_{11} compete. v_7 and v_8' win due to specificity. v_7 and v_8' also have same static priorities, but v_7 wins due to dynamic priority and the result is $(1, 1, 0, 0, 0, 1, 2_T)$. If a vector $(1, ?, 2, ?, 0, 0, ?_T)$ is presented, a similar situation would arise between v_{11} and v_{12}. The conflict would be resolved with static priorities.

3.4.6. Extensionality and Intensionality. Precepts may have a static priority associated with them. Both rules and precepts keep track of the number of examples they cover. Rules are the result of extensionality, while precepts are intensional. With this combination of the intensional and extensional approaches, FLARE can resolve conflicting defaults, like the Nixon Diamond [REI81], by either being told which default prevails (e.g., religious conviction is more important than political affiliation), or by computing a dynamic priority (see discussion in sections 3.3.3 and 3.4.4) from examples of Republican-Quakers.

Most learning systems are purely extensional, while most reasoning systems are purely intensional. It is our contention that a combination of the two approaches offers the maximum flexibility. On the one hand, extensionality accounts for the system's ability to adapt to its current environment, i.e., to be more autonomous. On the other hand, intensionality provides a mechanism by which the system can be taught, and thus does not have to unnecessarily suffer from poor or atypical learning environments.

In the context of reasoning, precepts provide a useful medium to encode certain first-order language statements (e.g., the rule base of an expert system), that can in turn be learned by FLARE (in the usual way), and later be used for reasoning purposes.

3.5. FLARE's Automatic Generation of Precepts

Section 3.4. introduced the notion of precepts, and argued their usefulness as learning biases and rule encoding (in the classical sense). Precepts may be obtained from the domain knowledge of a human teacher. In the context of the construction of expert systems, this process is called knowledge elicitation. In this section however, we focus on one mechanism for the automatic generation of precepts implemented in FLARE.

Precepts can be viewed as some instantiation of commonsense knowledge to a particular domain. Hence, if some commonsense knowledge is available, it may potentially be used to generate precepts. Several attempts have been made at formalizing commonsense knowledge [BRE91, LUK90], and a huge database of commonsense facts, called Cyc, is currently being compiled [LEN90]. With such facts expressed in first-order language, and an appropriate setting for instantiation, FLARE can generate precepts that can then be used as biases for inductive learning, or for further reasoning about the specific domain to which they apply.

Consider Example 4 from Section 3.1. Assume that the system is to inductively learn rules regarding the suitability of lenses for patients from a set of examples whose attributes include the patient's tear-production rate (tpr). The first two statements in Example 4 capture commonsense knowledge about eyes, the relationship between low tear-production rate and dryness, and the danger of placing objects (or things) in dry eyes. When instantiated with the third statement (i.e., lenses are things), the general knowledge may be used to produce a domain-dependent precept that states that if a patient has low tear-production rate, then he/she should not be fitted lenses. The precept in turn provides a useful bias to the system during induction.

The process of generating precepts (or instantiating common sense) is essentially one of acquiring common sense and reasoning from it. In general, it proceeds as follows.

Generate_precepts

1. Perform Learning using
   a. Available commonsense rules.
   b. Rules that establish the connection between the domain and common sense.
2. Perform Reasoning with
   a. The R-attributes asserted as given in the rules (all others are $*$).
   b. The T-attribute designated as the target.

The result of step 2 is a domain-dependent precept. The above algorithm uses mechanisms that are well defined in FLARE. Prior to any other learning, and if some commonsense knowledge is provided, FLARE always apply the Generate_precepts algorithm. In Example 4 of Section 3.1, the three statements are used in step 1, with the third statement establishing the needed connection. In step 2, the attribute lenses is asserted to 1 (i.e., true), the attribute Tpr is asserted to low, and the attribute Fit is designated as the target.

Step 2 of the Generate_precepts algorithm relies on a teacher setting up the correct attributes. More automatic mechanisms could be used, where the system could try any combination of the learning problem's R-attributes values to instantiate commonsense knowledge. Then, any such instantation that causes the T-attribute to become asserted is a potential precept. Of course, that process is exponential, and most of it would probably not lead to any useful conclusion.

4. Simulation Results

A set of classical commonsense benchmark problems is proposed in [LIF88], and [MUR92] contains many useful training sets for inductive learning. This section reports results of simulations of FLARE on several of these datasets.

One artifact of the implementation is that since variables cannot be dynamically added, all attributes must be defined *a priori*. All attributes that do not appear in rules, examples, or precepts are set to don't-care. This is consistent with the semantics of don't-care, and does not interfere with the algorithm since the distance *D* essentially treats learned don't-cares as neutral values.

4.1. Inductive Learning and Prior Knowledge

Several datasets from [MUR92] were chosen. They represent a wide variety of situations, as shown in Table 2. The column Size indicates the total number of examples in the dataset. The column Attributes records the number and type (L for linear, N for nominal) of all the attributes, other than the target (or output) attribute. The column Output shows the number of output classes.

Table 2 - Selected Applications

Application	Size	Attributes	Output
lenses	24	1L/3N	3
voting-84	435	16N	2
tic-tac-toe	958	9N	2
hepatitis	155	6L/13N	2
zoo	90	16N	7
iris	150	4L	3
soybean-small	47	4L/31N	4
segmentation	420	19L	7
glass	214	9L	7
breast-cancer	699	9L	2
sonar	208	60L	2
LED	1000	7N	10

FLARE's results were gathered for each of the above applications. They are shown in Table 3. Each of the reported results is an average over 10 runs of FLARE. The dataset is split into two halves, one of which, the training set, is used for learning, and the other, the test set, only for reasoning. In each run, the training set and the test set are randomly regenerated. The first number (PA) represents predictive accuracy (in %) on the test set after training and the second number (IR) is the inductive ratio, defined as the ratio of the size (in number of rules) of the final knowledge base to the number of instances used in learning. IR is another measure of the generalization power of FLARE, as well as an indication of FLARE's memory requirements.

Table 3 - FLARE: Induction

Application	PA	IR
lenses	78.3	.58
voting-84	91.6	.68
tic-tac-toe	80.4	1.0
hepatitis	80.8	.97

zoo	95.3	.47
iris	95.1	.20
soybean-small	99.1	1.0
segmentation	92.2	.99
glass	58.9	.27
breast-cancer	95.8	.49
sonar	81.7	.83
LED	70.4	.15
Averages	**85.0**	**.64**

These results compare favorably with results reported for other inductive learning algorithms (see [AHA91, QUI86, WET94, ZAR94]). Note that LED is a noisy application, for which the Bayesian optimal is 74%. FLARE handles the noise well, both by achieving good predictive accuracy, and by constructing a significantly small network.

Simulations of the first five applications with precepts given *a priori* (before the training set is presented) are reported in Table 4. Each column shows both PA and IR. Here, the precepts are obtained from domain knowledge provided with the application (voting-84) or generated from the authors' common sense (zoo, lenses, hepatitis, tic-tac-toe). They serve as learning biases.

Table 4 - FLARE: Induction and Prior Knowledge

Application	no prec.	w/prec.
lenses	78.3 - .58	81.7 - .42
voting-84	91.6 - .68	94.9 - .28
tic-tac-toe	80.4 - 1.0	87.7 - .72
hepatitis	80.8 - .97	82.2 - .66
zoo	95.3 - .47	95.3 - .42
Averages	**85.3 - .74**	**88.4 - .50**

The results with precepts show an average increase of 3.6% in predictive accuracy, and a decrease of 32.4% of IR (i.e., the resulting networks are significantly smaller on average).

4.2. Classical Commonsense Protocols

Several problems from the set of Benchmark Problems for Formal Nonmonotonic Reasoning [LIF88], were presented to FLARE. The problems were first translated into their corresponding AVL representation. FLARE is able to properly incorporate the premises, and correctly derive the expected conclusions for the following classes of problems from [LIF88]:

- A1 - basic default reasoning[8]

[8] A1's problem statement includes implications as well as facts. It is given as follows: blocks A and B are heavy, heavy blocks are normally located on the table, A is not on the table. Translating to AVL gives: A 1_T *, B 1_T *, * 1 1_T, A * 0_T. Now, if A ? $?_T$ is shown, A 1 $?_T$ will be derived from the first vector, and then A 1 $?_T$ matches exactly * 1 1_T and A * 0_T. Priority should be given to the later since it involves A

- A2 - default reasoning with irrelevant information
- A3 - default reasoning with several defaults
- A5 - default reasoning in an open domain
- A9 - priority between defaults
- B1 - linear inheritance (top-down)
- B2 - tree-structured inheritance
- B3 - one-step multiple inheritance
- B4 - multiple inheritance

Problem A4 involves a disabled default, and problems A6 through A8 deal with unknown exceptions. Such problems can not be represented in FLARE. Problems A10 and A11 deal with instances of defaults and reasoning about priority. Though not directly representable in FLARE, they are effectively solved via the use of static (A10) or dynamic (A11) priorities. The other classes of problems in [LIF88] (i.e., reasoning about actions, uniqueness of names, and autoepistemic reasoning) are beyond the current scope of FLARE.

In addition to the above protocols, another important commonsense protocol is one based on similarity, where if A implies B and C is similar to A, then instantiating C causes the assertion of B. This is exemplified by the Chaco example discussed earlier, and involves completion. With a value greater than 0 for the threshold T_D (see Section 3.3.5), FLARE correctly performs completion and forward chaining based on similarity to derive the proper conclusion.

The above problems are characteristics of important forms of human commonsense reasoning. However, they are artificial as they have been manufactured explicitly with the intent of isolating one salient feature of nonmonotonic reasoning, independent of all others. To further investigate the properties of FLARE and the combination of learning and reasoning, other more "real-world" applications must be designed and experimented with. The following section presents such preliminary applications.

5. Results on More Real-World Applications

In this section, several useful and more realistic applications are reported, that further exercise FLARE's abilities. Extensionality and the combination of learning with reasoning are of particular interest.

5.1. The Nixon Diamond

The Nixon Diamond [REI81], reproduced as Example 1 in Section 3.1, is important as a prototype of a class of interesting problems involving conflicting defaults. It is used here

---

explicitly. To solve this problem, vectors involving explicit references to instances of objects have their static priority set to 1 while all other vectors have their static priority set to 0. This is of course an artifact of encoding. An alternative is to write all facts relative to a given instance as a definition whose target is the instance value. Then completion would guarantee the correct outcome. The same applies to problems A2, A3, and A5.

to demonstrate FLARE's mechanisms to handle such conflicts both intensionally and extensionally.

FLARE's static priorities offer a simple way of resolving the Nixon Diamond intensionally, based on some externally provided information (e.g., religious convictions supersede political affiliations). In that case, both defaults are given along with an appropriate static priority.

Another alternative consists of providing the defaults without any priority. This corresponds to a possibly more natural situation where the system really is in a don't-know state when it comes to deciding on Nixon's dispositions. Yet, such don't-know states are uncomfortable and it is our contention that any kind of information that may allow a decision to be made should be used. Hence, we choose an epistemological approach, where the conflict arises due to beliefs rather than facts. In this case, it is possible to attempt to resolve the conflict by observing instances of Republican-Quaker. The relative number of Pacifist and non Pacifist could then serve as evidence to lean towards one decision or the other. In other words, it is the system's observation of what seems most common in its environment that creates its belief. This is not unlike the way humans deal with many similar situations.

A final approach, which combines inductive learning and reasoning, consists of not providing the system with any default. Rather, examples of Republicans, Quakers, and Republican-Quakers are shown and the system automatically comes up with both the defaults (through induction) and their relative priorities.

All three of these experiments were run with FLARE, and the results are as expected. In the third case, the actual knowledge base depends upon the ordering. It consists of one vector for Republicans that are not Quakers or one vector for Quakers that are not Republicans, one default vector for Quakers or Republicans, and one vector for Republican-Quakers. The target value of the vector for Republican-Quakers is not had via dyn_pty, but via the counters. Functionally, the result is identical.

5.2. Do Birds Typically Fly?

Incremental learning is one of FLARE's important features. With incrementality, the system is self-adaptive in the sense that its current knowledge base is representative of its experience with its environment so far. And the knowledge base can be continually updated as new information becomes available. To exercise incrementality a simple example of bottom-up inheritance involving birds was designed.

The application has four R-attributes, two of which correspond to Penguin and Bird. The other two are other (undetermined) attributes of birds (e.g., Feather). The T-attribute is Boolean and characterizes the ability to fly. At first, the system is exposed mostly to ostrich-birds (maybe the experiment is started in Australia). When asked whether birds typically fly (i.e., only the Bird attribute is asserted, and all other inputs are don't-know), FLARE concludes that birds do not fly, which is consistent with its current experience with the "world." However, as more new instances of flying birds (i.e., other than ostriches) are encountered, FLARE adapts its knowledge and when asked again, concludes that birds fly.

Of course, a precept may also be given to the system at any given time, stating that birds typically fly. The idea is that FLARE offers both options naturally. The system may be taught so as not to suffer from poor or atypical learning environments (e.g., Australia for birds' flying ability prediction), or it may be left to adapt to its environment. As research on autonomous agents continues, this later ability becomes very attractive and useful.

5.3. Learning Expert Systems

In order to better assess FLARE's reasoning mechanisms, two expert systems knowledge bases were used. One is called *mediadv* [HAR85] and is intended to help designers or committees choose the most appropriate media to deliver a training program. It consists of 20 rules with chains of inference of length at most 2. The other is called *health* [SAW86] and is intended to predict the longevity of patients based on a variety of factors (e.g., weight, personality, etc.). It is much larger as it contains 77 rules, and more complex as it involves longer chains of inference. Five rules were left out as one is redundant (i.e., rule 17 is identical to rule 14) and four are only needed in the interactive setting in which the original system is described. Hence, only 72 rules are considered.

Both sets of rules were translated into AVL. The 20 rules of *mediadv* produce 99 vectors, and the 72 rules of *health* produce 72 vectors. The number of vectors for *mediadv* is much larger than the original number of rules because many of the rules contain internal disjunctions. In AVL, a new vector must be constructed for each possible combination arising from the disjunctions. For example, the rule: If ((A=1 or A=2 or A=5) and (B=2.9 or B=7.8)) then C, gives rise to 6 vectors corresponding to the equivalent set of rules: If ((A=1) and (B=2.9)) then C, If ((A=1) and (B=7.8)) then C, If ((A=2) and (B=2.9)) then C, etc.

The sets of vectors corresponding to the original knowledge bases are not encoded into FLARE. Rather, they are presented to the system to be learned (i.e., the algorithm Learning is applied to them). Hence, some generalization may take place. In fact, the final number of vectors (after learning) in *mediadv* is only 71, while in *health* it is 65.

FLARE seems to perform reasonably well when asked to reason about its acquired knowledge. The matrix analysis of *mediadv* in [HAR85] was used to derive the 46 possible situations covered by the 20 rules in the knowledge base. After learning the rules, FLARE was exposed to these 46 situations and derived the proper conclusion for all except three. The three situations missed correspond to pairs of situations having identical premises, but different conclusions, and result from the inherent inconsistency (conflict between rules 13 and 14, and rules 16 and 17) in the knowledge base. In other words, there are only 43 distinct situations. When a precedence scheme is selected, and one vector from each pair is removed, FLARE derives the proper conclusion for all remaining 43 situations. This result is significant here because FLARE generalizes the original knowledge base (from 99 to 71 vectors). Hence, the knowledge base learned by FLARE may not be logically equivalent to the original one. It appears however that, at least in the case of *mediadv*, the system is logically equivalent and also more parsimonious. A similar experiment is difficult to conduct for the *health* knowledge base as the number of possible situations it covers is much larger. A formal investigation of FLARE's generalization scheme is necessary to provide a result on logical equivalence in the general case.

Further representative experiments on new situations (i.e., not explicitly encoded in the knowledge bases) were performed. They are summarized in Table 5 and Table 6. In each table the first column contains the list of attributes used in the knowledge base. Then, each pair (setting, result) of columns represents an experiment in reasoning with the knowledge base. The setting column contains the data FLARE starts with. Unknown conditions (or attributes) are initialized as don't-knows (i.e., ?). The result column shows the state of knowledge after reasoning. Each derived piece of information is italicized and subscripted by the depth of inference at which it was derived.

Table 5 - Mediadv Knowledge Base

	setting 1	**result 1**	**setting 2**	**result 2**	**setting 3**	**result 3**	**setting 4**	**result 4**
situation	view	view	diagram	diagram	object	object	diagram	diagram
stim-sit.	?	*pictorial*$_1$	?	*pictorial*$_1$	?	*environ.*$_1$	?	*selective*$_1$
response	building	building	moving	moving	?	?	matching	matching
app-resp.	?	*motor*$_1$	?	*motor*$_1$	?	?	?	
stim-dur.	persist.	persist.	brief	brief	?	?	brief	brief
feedback	yes	yes	yes	yes	?	?	yes	yes
pr-mod.	response	response	response	response	?	?	response	response
budget	medium	medium	medium	medium	?	?	small	small
media	?	*self-study*$_2$	?	*tutor*$_2$	?	*lab*$_2$	?	*tutor*$_2$

The first setting matches both rules 11 and 20, so one of them is chosen at random. In the second setting, the change in duration from persistent to brief suffices to bring the situation closer to rule 20. The third setting tests the system in cases where very little information is available, but that information is relevant. Only one rule contains a premise for stimulus-situation equal to environmental (rule 7). It is therefore applied. The fourth setting was designed to reflect a pattern recognition scenario that may be used in a first-grade classroom, where the children would be shown various shapes and their names, and then asked to classify new objects in terms of their shapes. In this case, the system's prediction seems to match intuition. It suggests the use of a human tutor to perform the training. Notice that this last setting and its outcome give support to the (now classical) notion of supervised learning.

Table 6 - Health Knowledge Base

	setting 1	**result 1**	**setting 2**	**result 2**	**setting 3**	**result 3**
rel. weight	?	*normal*$_1$	?	*obese*$_1$	?	*normal*$_1$
val	?	*yes*$_2$	?	?	?	*yes*$_2$
heart-dis-risk	?	?	?	*>average*$_1$	?	*average*$_1$
hddanger	?	?	?	?	?	*low*$_2$
start	yes	yes	yes	yes	yes	yes
age	25-55	25-55	>55	>55	<25	<25
gender	F	F	M	M	F	F

base-longevity	?	*67*$_1$	?	*60*$_1$	?	*72*$_1$
weight	<110	<110	>170	>170	110-170	110-170
frame	small	small	small	small	large	large
cholesterol	?	?	high	high	low	low
fat intake	normal	normal	high	high	normal	normal
salt intake	normal	normal	high	high	normal	normal
blood-pressure	?	*average*$_1$	?	*>average*$_1$	?	*average*$_1$
calcium	?	?	?	?	normal	normal
osteoporo-risk	?	?	?	?	?	*average*$_1$
smoker	no	no	yes	yes	no	no
outlook	?	?	?	*bleak*$_2$	?	*fair*$_3$
race	asian	asian	caucasian	caucasian	caucasian	caucasian
origin	?	?	medit.	medit.	n-amer.	n-amer.
risk	?	*high*$_2$	?	*high*$_2$	unknown	unknown
personality	aggressive	aggressive	aggressive	aggressive	docile	docile
person. type	?	*type_a*$_1$	?	*type_a*$_1$	?	*type_b*$_1$
alcohol cons.	moderate	moderate	excessive	excessive	none	none
add	?	*good*$_1$	?	*poor*$_1$	?	*fair*$_1$
factor	?	*none*$_3$	?	*minus_12*$_3$	?	*plus_12*$_4$
longevity	?	*67*$_4$	?	*48*$_4$	?	*84*$_5$

Notice the greater length in the chains of inference compared to *mediadv*. The first setting corresponds to an average individual, with little vices or excesses. As one would expect, its longevity is average. The second setting corresponds to a very unhealthy individual which one would not expect to live very long. FLARE concludes accordingly. The third setting corresponds to a healthy female individual which one would expect to live a good number of years, as predicted by the system. Another setting, not shown here, was used in which only the value of start was set to "no." As expected, the system concludes with longevity value "quit."

Note that as in classical expert systems, the identification of a closest match during reasoning can be used to query the user for missing information as well as to justify both the queries and the decisions made. Of interest also is the case of conflicts. As in the previous examples, learning can beneficially be combined with reasoning. For example, in *mediadv*, the following rules are found, where X is some fixed conjunction of conditions.

- rule 13: if (X) and (training_budget = small or training_budget = medium) then
  media_to_consider = lecture
- rule 14: if (X) and (training_budget = medium) then
  media_to_consider = lecture-with-slides

It is clear that in some cases, these rules conflict. It is difficult to avoid such occurrences in large knowledge bases elicited form experts. As FLARE supports learning, it is possible however to look at various (historical) situations where training_budget was medium, and which media was used. This information can in turn be used to give precedence to one rule over the other. Moreover, this precedence need not be fixed after so many examples have been considered. Indeed, it may evolve over time, and even change radically depending on circumstances. Alternatively, static priorities allow

An example was implemented in which several instances of [(X) and (training_budget = medium)] together with a target value for media_to_consider. The instances used cause the value of dyn_pty of rule 13 to be greater than that of rule 14, thus effectively giving (evidential) precedence to rule 13.

5.4. Generating Precepts

Section 3.5 discussed the possibility of using reasoning to instantiate some commonsense knowledge to a particular domain, thus producing domain-dependent precepts that can in turn be used as learning biases during induction.

Example 4 from Section 3.1 was implemented (as described in Section 3.5) and a precept stating that if the Tpr attribute is set to low, then lenses should not be prescribed was generated. That precept was in turn used prior to performing inductive learning on the lenses dataset of Section 4.1.

The above process is two-phase, where both phases perform the same operations on different pieces of information. In the first phase, commonsense knowledge expressed in first-order logic (and translated into AVL) is learned by FLARE. Then FLARE reasons based on some instantiation that links the general commonsense knowledge to the current domain. The result of this reasoning phase is one (or more) vector containing domain-dependent information. In the second phase, FLARE learns from the generated domain-dependent information and any other available examples. The result is a set of inductively generated rules.

5.5. Further Experiments

The above applications serve to demonstrate that FLARE holds promise. Further usage of the proposed framework will be developed. In particular, work must be pursued in finding ways to reason about inductively generated rules. For example, in the Chaco example, the definitions used were given *a priori*, rather than learned by induction. Several questions remain open.

- Will inductively learned rules always be good for reasoning?
- What forms of reasoning would they support?
- How do we deal with disjunctive representations?
- Where do we find applications of such systems (like the UC Irvine repository)?

Another interesting avenue to pursue is the discovery of mechanisms to use the ability to reason as a guide to further learning. For example, is it possible to generate other types of biases than precepts? Or, could the identification of a conflict (such as the one discussed in Section 5.3) be used to determine what kinds of examples to seek?

6. Conclusion

This paper introduces a model that attempts to combine inductive learning with prior knowledge and reasoning into a unified framework. The system, called FLARE, is detailed. Its reasoning mechanism is a combination of rule-based reasoning and similarity-based reasoning, which reduces the susceptibility to brittleness. Reasoning is inherently nonmonotonic and accounts for many useful commonsense protocols. Learning is incremental and follows a form of nearest-hyperplane learning. Results on several real-world datasets compare favorably with other similar inductive learning systems.

Several applications of the framework are presented, including the automatic generation of precepts to bias inductive learning and the use of examples to resolve conflicting defaults extensionally. Current results demonstrate promise. Future work includes the following:

- Finding mechanisms to use reasoning to guide learning
- Attempting to overcome (or appropriately use) the order-dependency
- Providing support for internal disjunction
- Using inductively learned rules in reasoning (the support is available, but the induction may not produce useful rules)
- Possibly incorporating backward chaining
- Translating system's knowledge base back from AVL to FOL
- Further experimenting with larger applications

CHAPTER 9
Conclusion

In this concluding chapter, we briefly summarize the results of this dissertation and discuss potentially promising avenues of future research.

1. Summary

The aim of this dissertation was to develop integrated frameworks for learning and reasoning. We first proposed a set of seven desirable properties for such frameworks. These properties serve as the foundation to the design of the various models we have proposed. The first model, PDL2, was an extension of previous work and focused on simple commonsense reasoning protocols. Then, we developed a heterogeneous metric whose objective is to improve predictive accuracy and widen the range of applications effectively handled by our systems. An extension of that metric was used in our next model, ILA, whose focus was on improved inductive learning abilities (speed and predictive accuracy), and on incrementality. Finally, we presented a more general framework, called FLARE, whose focus was on increased reasoning power, and greater, more meaningful integration of learning and reasoning. This framework supports forward chaining, and the automatic generation of precepts from commonsense knoweldge expressed in first-order logic clauses. Several useful examples were also introduced that serve as benchmark.

The proposed systems perform inductive learning in an incremental fashion and generate rules that represent hyperplanes in the input space. Prior knowledge is in the form of precepts which are also hyperplanes in the input space. The reasoning mechanisms are a combination of rule-based (or default) reasoning and similarity-based reasoning. The proposed models have been tested empirically against several real-world applications and results demonstrate promise. Predictive accuracy compares favorably with other inductive models [ZAR94], and many useful commonsense protocols are accounted for both intensionally and extensionally.

2. Future Research

The problem of integrating learning and reasoning is complex. This dissertation has not resolved it. We have only proposed several models that attempt to construct such a unified framework. The models have several limitations. In this section, we examine some of these limitations with the intent of uncovering useful avenues of future research. We look at specific extensions to the proposed models, discuss the question of scaling up, and finally re-visit the issue of common sense.

2.1. Extensions to the Proposed Models

The work of Chapter 5 resulted in a heterogeneous metric capable of handling don't-cares and missing attributes. However, the metric assumes that the range of each attribute is known *a priori*, which goes somewhat contrary to the design of incremental learning systems. A more adaptive mechanism, such as an incremental standard deviation, could be used instead of the range for purposes of normalization.

Though ILA and FLARE handle continuous-valued attributes directly, a better treatment of such attributes is still necessary. As pointed out in [BER94], actual continuous-valued attributes appear to be rarely used directly in human decision making. Rather, they are translated into a coarse set of discrete values, and it is those discrete values that are used in the learning system. The proposal in [BER94] provides a good starting point, particularly as it is context-dependent, rather than solely driven by statistical considerations. Discretization of ordered (e.g., continuous) values does not affect ordering so that the metric we designed is still useful.

The inductive power of the models proposed in this dissertation compares favorably (at least in terms of predictive accuracy) with that of other inductive systems. However, the dropping the condition generalization mechanism is still very primitive. It confines all generalizations to be hyperplanes. Also, its implementation makes no sense on continuous-valued attributes. Rather than dropping the condition altogether, and thus producing a generalization that covers the full range of possible values for a nominal attribute, it would be interesting to try the use of ranges or even sets of values as generalizations.

The representation of concepts (or classes) by the current systems consists of disjunctive conjunctions. That is, the systems are likely not to generate a single conjunction for a concept, but rather a set of such conjunctions. Though this may tend to improve predictive accuracy (by producing a finer decomposition of the input space), it makes the application of the completion principle (in FLARE) more challenging, and possibly too strong as concepts are not defined tightly enough.

Only one form of prior knowledge is used to bias learning, namely precepts, which are also hyperplanes in the input space. Another form of prior knowledge used is that embedded in the first-order formulation of (potentially) available commonsense rules. This latter form of knowledge is not explicit, but it may be instantiated to produce precepts applicable to a specific domain. Other forms of prior knowledge exist and should be investigated. They may not however fit as naturally into the systems presented here.

None of the proposed models support the dynamic addition of attributes. All attributes must be specified *a priori*. This runs counter to the notion of incrementality and to nature. The fact is that we, as humans, often do encounter new things. The ability to add new attributes "on-the-fly" would thus increase the flexibility of the system, and make it inherently more self-organizing and autonomous.

Though several mechanisms have been implemented to limit the effects of ordering, the models are still order-dependent. This could however be overcome by using memory. Humans do store information so that reordering is possible if necessary.

The systems lack the elegance of the localist/distributed (or conceptual/subconceptual) representation dichotomy of CONSYDERR [SUN92]. Both the concept and its attributes are encoded at the same level, in a vector. The difficulty is that the same attribute may be used as a predictive (subconceptual) attribute in a given context (e.g., to learn or build another concept), and as a target (conceptual) attribute in another context (e.g., discover its critical features). Of course, this problem can be carried out to arbitrary depths. Of the proposed models, FLARE is the only one that attempts to provide a solution. Because FLARE is able to identify targets and to subsequently deal with vectors of different targets, it can essentially implement any multi-level structure as a "planar" projection. The price to pay is a somewhat more cumbersome algorithm. An advantage of FLARE over CONSYDERR is that the model can be instantiated both at the conceptual and the subconceptual levels. A disadvantage is the lack of an explicit mechanism to capture the semantical differences between levels. Further research in this area may prove useful.

There is no mechanisms provided for variable binding in any of the proposed models. At this stage it is not clear whether this is a critical deficiency of the systems. More work needs to be done to discover the implication of such a limitation and to find effective ways to overcome it, if necessary.

Finally, further research must be devoted to a more complete integration of learning and reasoning. Of particular interest are issues such as backward chaining, the use of reasoning to guide learning, and the use of inductively learned rules to reason. As these new issues are addressed and solutions are proposed, the need for more (real-world) benchmark applications that test the solutions' implementation naturally arises.

2.2. Scaling Up: Some Thoughts on Cyc

It has already been noted that the aim of the models presented here is to integrate learning and reasoning into a unified framework. In effect, this work attempts to bridge the gap between a large body of research in inductive learning and another large body of research in reasoning. Many datasets, knowledge bases, problems, and applications have been designed to test each approach separately, but not concurrently. The examples used in this dissertation are among the first ones designed to test integrated frameworks. They are still very limited however, and relatively small.

The systems presented in this dissertation have been shown to perform well on these simple problems. It is reasonable to ask what happens when the problems become more complex, i.e., of the type and size humans deal with on a regular basis. Though it is tempting to say that the mechanisms are in place for the system to effectively scale up, empirical tests on large, real-world problems should be conducted.

The work on Cyc [LEN90, LEN94] thus becomes very relevant to the present research. One of the aims of the Cyc project is to capture most of human commonsense knowledge in a huge database that can be used by a variety of models. The knowledge base is partitioned into micro-theories or contexts. It is reasonable to envision testing FLARE on one (or more) such context. The commonsense facts and rules available in the Cyc's context would serve as the basis for reasoning, and examples drawn from experience would be used to adapt to a particular situation.

Moreover, a system for inferences has also been developed for the Cyc knowledge base. It would be useful to compare FLARE's reasoning mechanisms with those of Cyc. The decade-long experience of the designers of Cyc may shed some light on our current efforts and potentially uncover both promising directions for future work and dead ends. For example, it appears that the use of purely numerical confidence levels for conflict resolution between rules is not satisfactory for a very large set of rules [LEN94]. It is suggested that explicit rules relative to the conflicts (e.g., P is more likely than Q) be used to reason about the conflicts. However, Cyc's knowledge base is currently being programmed by humans, so that rules are explicitly taught. In FLARE, some rules are learned inductively, and certain conflicts are resolved through the use of evidence gathered from experience. Currently, numerical values (dynamic priority, counters, etc.) are used to gather that evidence. It is should be possible to translate the information provided by the numbers into rules. FLARE would then have to be extended to reason about rules, and to carry out uncertain inferences.

Finally, FLARE may offer a more synergistic way to achieve the goals of Cyc, since FLARE both learns and reasons. The Cyc's knowledge base is programmed by humans. Alternatively, it is conceivable that the knowledge could be had inductively as suggested in [MIN94]. The task is daunting but fits within FLARE's paradigm. FLARE could be taught some things by a human teacher. It could also learn from its own experience. The human teacher in turn could also learn from feedback received from FLARE, and thus guide the system further. This approach naturally extends to the more general notion of intelligent agents.

2.3. Common Sense Revisited

One of our main focuses on reasoning has been to capture *commonsense* reasoning patterns in our models. Common sense can be defined as expertise about the everyday world [WOL94]. There are several ways to model how such expertise may be acquired and used for reasoning. The work of this dissertation is based exclusively on the view expressed in [SUN92], where commonsense reasoning is characterized as a combination of rule-based reasoning and similarity-based reasoning.

One related view expressed in [WOL94] considers that information in many situated cases (i.e., well-defined, contextual situations) is needed to overcome the frame problem (i.e., to handle common sense). Hence, it follows that case-based (or memory-based) reasoning best satisfies the characteristics of commonsense reasoning. Because the retrieving of cases in case-based reasoning is based on some notion of similarity, this approach is similar to the one we used.

The main advantage of these two views is that they lend themselves naturally to learning. Either the rules or the cases do not have to be pre-encoded. They may indeed be learned inductively from experience.

There are however other ways to devise a solution. We briefly suggest one here that we are currently working on [GIR94f]. It is based on object-oriented analysis, and more particularly on the Object-Oriented Systems Analysis (OSA) model [EMB92]. Inductive learning often consists of coming up with classification rules for concepts. This process is very similar to the process of constructing object classes based on sets of common properties. The membership conditions of an object class are homologous to the critical features of a learned concept. Typically, the OSA approach is intensional however. Provision could be made to support some extensional learning, by for example recording instances of relationships between objects. If sufficiently many relationships exist, a relationship set may be "induced" between the corresponding classes of objects. One interesting learning model from relational databases is in [CAI90]. Another incremental learning algorithms based on Galois lattices is in [GOD91].

A further advantage is the support for arbitrary relationships as well as, in particular, generalization. The problem of inheritance becomes one of graph traversal and conflicting defaults can be handled by choosing the path of shortest length. Mechanisms have been designed, based on Galois lattices, to inductively learn inheritance hierarchies [GOD93].

Moreover, as the system supports both capabilities and common properties, distinction can be made between material implications and evidential implications (defaults). As OSA also includes a sophisticated behavior model, it is also possible to use the system for temporal reasoning. It appears that state nets may indeed offer natural solutions to several challenging problems in this area.

Finally, one important aspect of common sense is the natural abiltiy we have to predict others' behavior and reactions [FOR89]. We would argue that this is due to the fact that we share the same underlying conceptual model of the world. If our meta-models are similar, I can represent what I know, I can represent what you know, I can "compute" what I would do and thus infer what you are most likely to do. The Object Relationship Model (ORM) of OSA is a useful tool to capture such a meta-model and reason about it. In this context, commonsense reasoning can be characterized as a form of analogical reasoning based on instantiations of the meta-model. Note that the idea that people, or more generally intelligent agents, must share a vast body of knowledge in order to be competent is the underlying motivation of the Cyc project [LEN94].

An approach based on semantic nets (of which ORMs are a special form) is also presented in [SHR87, SHG89]. The model, called SNePS, is formalized and defines a non-monotonic logic whose syntax closely matches natural language and whose (intensional) semantics handle many commonsense protocols. The basic model has been extended to reason about actions and to put acting in the service of inference [KUD93, KUD94]. This extension may be useful to the design of autonomous agents, though it is still limited to an intensional semantic (i.e., no induction).

BIBLIOGRAPHY

[AHA91a] Aha, D.W. (1991). A Study of Instance-Based Algorithms for Supervised Learning Tasks. Technical Report, University of California, Irvine.

[AHA91b] Aha, D.W., Kibler, D., and Albert, M.K. (1991). Instance-Based Learning Algorithms. *Machine Learning*, 6, 37-66.

[BER94] Bertelsen, R. (1994). Extending ID3 Through Discretization of Continuous Inputs. In *Proceedings of the Seventh Florida AI Research Symposium (FLAIRS'94)*, 122-125.

[BLU87] Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M.K. (1987). Occam's Razor. *Information Processing Letters*, 24, 377-380.

[BRE91] Brewka, G. (1991). *Nonmonotonic Reasoning: Logical Foundations of Commonsense*. Cambridge University Press.

[BUN90] Buntine, W.L. (1990). A Theory of Learning Classification Rules. Ph.D. Dissertation, University of Technology, School of Computing Science, Sydney.

[CAI90] Cai, Y., Cercone, N., and Han, J. (1990). An Attribute-Oriented Approach for Learning Classification Rules from Relational Databases. In *Proceedings of the Sixth International Conference on Data Engineering*, 281-288.

[CAR83] Carbonell, J.G. (1983). Learning by Analogy: Formulating and Generalizing Plans from Past Experience. In Michalski, R.S., Carbonell, J.G., and Mitchell, T.M., (Eds.). *Machine Learning: An Artificial Intelligence Approach*. Tioga Publishing Company, Palo Alto, CA, Chapter 5.

[CHE91] Cheng, Y. (1991). Context-Dependent Similarity. *Uncertainty in Artificial Intelligence*, 6, 41-47.

[CLB84] Clancey, B.C., and Shortliffe, E.H. (1984), Editors. *Readings in Medical Artificial Intelligence: The First Decade*. Reading, MA: Addison-Wesley Publishing Company.

[CLK78] Clark, K.L. (1978). Negation As Failure. In Gallaire, H., and Minker, J., (Eds.). *Logic and Databases*. Plenum Press, 293-322.

[CLP89] Clark, P., and Niblett, T. (1989). The CN2 Induction Algorithm. *Machine Learning*, 3, 261-283.

[COL89] Collins, A., and Michalski, R.S. (1989). The Logic of Plausible Reasoning: A Core Theory. *Cognitive Science*, 13, 1-49.

[COR84] Cordier, M.O. (1984). Les Systems Experts. *La Recherche*, Vol. 15, No. 151, 60-69. (in French).

[DAS91] Dasarathy, B.V. (1991). *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press.

[DAV90] Davis, E. (1990). *Representations of Commonsense Knowledge*. Morgan Kaufmann Publishers, Inc., San Mateo, CA.

[DES93] Desjardins, M. (1993). Workshop Summary: AAAI'92: Workshop on Constraining Learning with Prior Knowledge. *SIGART Bulletin*, Vol. 4, No. 3, 11-13.

[DIE83] Dietterich, T.G., and Michalski, R.S. (1983). A Comparative Review of Selected Methods for Learning from Examples. In Michalski, R.S., Carbonell, J.G., and Mitchell, T.M., (Eds.). *Machine Learning: An Artificial Intelligence Approach*. Tioga Publishing Company, Palo Alto, CA, Chapter 3.

[DIG84] D'Ignazio, F., and Wold, A.L. (1984). *The Science of Artificial Intelligence*. Franklin Watts Library Edition.

[DRE79] Dreyfus, H. (1979). *What Computers Can't Do: The Limits of Artificial Intelligence*. Harper & Row, Publishers, Revised Edition.

[DRE92] Dreyfus, H. (1992). *What Computers Still Can't Do: A Critique of Artificial Reason*. MIT Press.

[DUD84] Duda, R.O., and Reboh, R. (1984). AI and Decision Making: The PROSPECTOR Experience. In Reitman, W., (Ed.). *Artificial Intelligence Applications for Business*. Norwood, NJ: Ablex Publishing Corp.

[ELM91] Elman, J.L. (1991). Incremental learning, or The importance of starting small. CRL Technical Report 9101, La Jolla, CA: University of California, San Diego, Center for Research in Language.

[EMB92] Embley, D., Kurtz, B., and Woodfield, S.N. (1992). *Object-Oriented Systems Analysis: A Model-Driven Approach*. Prentice Hall, Inc., NJ.

[FOR89] Forguson, L. (1989). *Common Sense*. Routledge.

[GEN89] Gentner, D. (1989). The Mechanisms of Analogical Reasoning. In Vosniadou, S., and Ortony, A., (Eds.). *Similarity and Analogical Reasoning*. London: Cambridge University Press.

[GIN87] Ginsberg, M.L. (1987), Editor. *Readings in Nonmonotonic Reasoning*. Morgan Kaufmann Publishers, Inc., Los Altos, CA.

[GIR92] Giraud-Carrier, C., and Martinez, T. (1992). Analysis of the Convergence and Generalization of AA1. To appear in *Journal of Parallel and Distributed Computing*.

[GIR93a] Giraud-Carrier, C. (1993). A Precept-Driven Learning Algorithm. Master's Thesis. Brigham Young University, Department of Computer Science.

[GIR93b] Giraud-Carrier, C., and Martinez, T.R. (1993). Using Precepts to Augment Training Set Learning. In *Proceedings of the First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems (ANNES'93)*, 46-51.

[GIR94a] Giraud-Carrier, C., and Martinez, T.R. (1994). Seven Desirable Properties for Artificial Learning Systems. In *Proceedings of the Seventh Florida AI Research Symposium (FLAIRS'94)*, 16-20.

[GIR94b] Giraud-Carrier, C., and Martinez, T.R. (1994). An Incremental Learning Model for Commonsense Reasoning. To appear in *Proceedings of the Seventh International Symposium on Artificial Intelligence (ISAI'94)*.

[GIR94c] Giraud-Carrier C., and Martinez, T.R. (1994). An Efficient Metric for Heterogeneous Inductive Learning Applications in the Attribute-Value Language. To appear in *Proceedings of the Third Golden West International Conference on Intelligent Systems (GWIC'94)*.

[GIR94d] Giraud-Carrier, C., and Martinez, T.R. (1994). ILA: Combining Inductive Learning with Prior Knowledge and Reasoning. Submitted.

[GIR94e] Giraud-Carrier, C., and Martinez, T. (1994). An Integrated Framework for Learning and Reasoning. Submitted.

[GIR94f] Giraud-Carrier, C. (1994). Non-Monotonic Inheritance and Temporal Reasoning with OSA. In preparation.

[GOD91] Godin, R., Missaoui, R., and Alaoui, H. (1991). Learning Algorithms using a Galois Lattice Structure. In *Proceedings of the Third IEEE International Conference on Tools for Artificial Intelligence*, 22-29.

[GOD93] Godin, R., and Mili, H. (1993). Building and Maintaining Analysis-Level Class Hierarchies Using Galois Lattices. In *Proceedings of OOPSLA'93*, 394-410.

[GOL93] Goldszmidt, M., Morris, P., and Pearl, J. (1993). A Maximum Entropy Approach to Nonmonotonic Reasoning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15, No. 3, 220-232.

[GUH94] Guha, R.V., and Lenat, D.B. (1994). Enabling Agents to Work Together. *Communications of the ACM*, Vol. 37, No. 7, 126-142.

[HAA83] Haas, N., and Hendrix, G.G. (1983). Learning by Being Told: Acquiring Knowledge for Information Management. In Michalski, R.S., Carbonell, J.G., and Mitchell, T.M., (Eds.). *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufmann Publishers, Inc., Vol. I, Chapter 13.

[HAL90] Hall, L.O., and Romaniuk, S.G. (1990). A Hybrid Connectionist, Symbolic Learning System. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 783-788.

[HAM89] Hammond, K.J. (1989). CHEF. In Reisbeck, C., and Schank, R., (Eds.). *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, Hillsdale, NJ.

[HAR85] Harmon, P., and King, D. (1985). *Expert Systems*. John Wiley & Sons, Inc.

[HAZ94] Hamilton, H.J., and Zhang, J. (1994). Learning Pronunciation Rules for English Graphemes Using the Version Space Algorithm. In *Proceedings of the Seventh Florida AI Research Symposium (FLAIRS'94)*, 76-80.

[HOB85] Hobbs, J.R., and Moore, R.C., (1985), Editors.. *Formal Theories of the Commonsense World*. Ablex Publishing Corporation.

[HOR90] Horty, J.F., Thomason, R.H., and Touretzky, D.S. (1990). A Skeptical Theory of Inheritance in Nonmonotonic Networks. *Artificial Intelligence*, 42, 311-348.

[KAE93] Kaelbling, L.P. Reinforcement Learning. (1993). *MLnet Workshop "Learning in Autonomous Agents,"* invited talk.

[KIB87] Kibler, D., and Aha, D.W. (1987). Learning Representative Exemplars of Concepts: An Initial Case Study. In *Proceedings of the Fourth International Workshop on Machine Learning*, 24-30.

[KON88] Konolige, K. (1988). Hierarchic Autoepistemic Theories for Nonmonotonic Reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI'88)*, 439-443.

[KUD93] Kumar, D. (1993). From Beliefs and Goals to Intentions and Actions - An Amalgamated Model of Acting and Inference. Ph.D. Thesis, State University of New York, Buffalo, NY.

[KUD94] Kumar, D. (1994). The OK BDI Architecture. In *Proceedings of the Third Golden West International Conference on Intelligent Systems (GWIC94)*. Forthcoming.

[KUN94] Kundu, S. (1994). The Completion Principle: A Technique for Discovering New Concepts in Inductive Learning. In *Proceedings of the Seventh Florida AI Research Symposium (FLAIRS'94)*, 126-130.

[KUV94] Kumar, V.R. (1994). Performance Comparison of Different Learning Methods for Weather Forecasting Operations. To appear in *Proceedings of the Third Golden West International Conference on Intelligent Systems (GWIC'94)*.

[LEC89] le Cun, Yann. (1989). Generalization and Network Design Strategies. In Pfeifer, R., Schreter, Z., Fogelman-Soulié, F., and Steels, L. (Eds.). *Connectionism in Perspective*. Elsevier Science Publishers B.V.

[LEN90] Lenat, D.B., et al. (1990). Cyc: Toward Programs With Common Sense. *Communications of the ACM*, Vol. 33, No. 8, 30-49.

[LER92] Lerher, L., and Kaindl, H. (1992). Problems, Communication, and Common Sense. *SIGART Bulletin*, Vol. 3, No. 2, 15-19.

[LIF88] Lifschitz, V. (1988). Benchmark Problems for Formal Non-Monotonic Reasoning. In *Proceedings of the 2nd International Workshop on Non-Monotonic Reasoning*. Printed in Reinfrank, M., de Kleer, J., Ginsberg, M.L., and Sandewall, E., (Eds.). *LNCS*, Springer-Verlag, No. 346, 202-219.

[LIF90] Lifschitz, V., (1990), Editor. *Formalizing Common Sense: Papers by John McCarthy*. Ablex Publishing Corporation, Norwood, N.J.

[LUK90] Lukaszewicz, W. (1990). *Non-Monotonic Reasoning: Formalization of Commonsense Reasoning*. Ellis Horwood Limited.

[MAR86] Martinez, T.R. (1986). Adaptive Self-Organizing Networks. Ph.D. Thesis, UCLA Tech. Rep. - CSD 860093.

[MAR88] Martinez, T.R., and Vidal, J.J. (1988). Adaptive Parallel Logic Networks. *Journal of Parallel and Distributed Computing*, Vol. 5, No. 1, 26-58.

[MAR91a] Martinez, T.R., and Campbell, D.M. (1991). A Self-Adjusting Dynamic Logic Module. *Journal of Parallel and Distributed Computing*, Vol. 11, No. 4, 303-313.

[MAR91b] Martinez, T.R., and Campbell, D.M. (1991). A Self-Organizing Binary Decision Tree for Incrementally Defined Rule Based Systems. In *IEEE Transactions on System, Man, and Cybernetics*, Vol. 21, No. 5, 1231-1238.

[MAR93] Martinez, T.R., Barker, J.C., and Giraud-Carrier, C. (1993). A Generalizing Adaptive Discriminant Network. In *Proceedings of the World Congress on Neural Networks*, I-613-616.

[MAR94] Martinez, T.R., Hughes, B., and Campbell, D.M. (1994). Priority ASOCS. *Journal of Artificial Neural Networks*, 1(3), 403-429.

[MCC80] McCarthy, J. (1980). Circumscription: A Form of Nonmonotonic Reasoning. *Artificial Intelligence*, 13, 27-39.

[MCC84] McCarthy, J. (1984). Some Expert Systems Need Common Sense. In Pagels, H., (Ed.). *Computer Culture: The Scientific, Intellectual and Social Impact of the Computer*. Annals of the New York Academy of Sciences, Vol. 426, 129-137.

[MIC83a] Michalski, R.S. (1983). A Theory and Methodology of Inductive Learning. *Artificial Intelligence*, 20, 111-161.

[MIC83b] Michalski, R.S., Carbonell, J.G., and Mitchell, T.M., (1983), Editors. *Machine Learning: An Artificial Intelligence Approach*. Tioga Publishing Company, Palo Alto, CA.

[MIN67] Minsky, M., and Papert, S. (1967). *Perceptrons*. MIT Press, Cambridge, MA.

[MIN91] Minsky, M. (1991). Logical Vs. Analogical or Symbolic Vs. Connectionist or Neat Vs. Scruffy. *AI Magazine*, 12, 2, 34-51.

[MIN94] Minsky, M., and Riecken, D. (1994). A Conversation with Marvin Minsky About Agents. *Communications of the ACM*, Vol. 37, No. 7, 22-29.

[MIT80] Mitchell, T.M. (1980). The Need for Biases in Learning Generalizations. Technical Report No. CBM-TR-117, New Brunswick, NJ: Rutgers University, Department of Computer Science.

[MOO89] Mooney, R., Shavlik, J., Towell, G., and Gove, A. (1989). An Experimental Comparison of Symbolic and Connectionist Learning Algorithms. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 775-780.

[MUR92] Murphy, P.M., and Aha, D.W. (1992). *UCI Repository of machine learning databases*. Irvine, CA: University of California, Department of Information and Computer Science.

[QUI86] Quinlan, J.R. (1986). Inductive Learning of Decision Trees. *Machine Learning*, 1, 81-106.

[REI80] Reiter, R. (1980). A Logic for Default Reasoning. *Artificial Intelligence*, 13, 81-132.

[REI81] Reiter, R., and Criscuolo, G. (1981). On Interacting Defaults. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, 270-276.

[RIB75] Ribes, J.C., and Ribaud, F. (1975). *Le Dossier de l'Intelligence Artificielle*. Librairie Artheme Fayard. (in French).

[RIN79] Ringle, M.D., (1979), Editor. *Philosophical Perspectives in Artificial Intelligence*. Humanities Press.

[ROS58] Rosenblatt, E. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65, 386-408.

[RUD91a] Rudolph, G. (1991). A Location-Independent ASOCS Model. Master's Thesis, Brigham Young University, Department of Computer Science.

[RUD91b] Rudolph, G., and Martinez T.R. (1991). An Efficient Static Topology for Modeling ASOCS. In Kohonen, et al., (Eds.). *Artificial Neural Networks*. Elsevier Science Publishing, 729-734.

[RUM86] Rumelhart, D. and McClelland, J. (1986). *Parallel Distributed Processing*. MIT Press, Cambridge, MA, Vol. 1.

[RYC83] Rychener, M.D., (1983). The Instructible Production System: A Retrospective Analysis. In Michalski, R.S., Carbonell, J.G., and Mitchell, T.M., (Eds.). *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufmann Publishers, Inc., Vol. I, Chapter 14.

[SAL91] Salzberg, S. (1991). A Nearest Hyperrectangle Learning Method. *Machine Learning*, 6, 277-309.

[SAW86] Sawyer, B., and Foster, D.L. (1986). *Programming Expert Systems in Pascal*. John Wiley & Sins, Inc.

[SEJ87] Sejnowski, T.J., and Rosenberg, C.R. (1987). Parallel Networks that Learn to Pronounce English Text. *Complex Systems*, 1, 145-168.

[SHA90] Shavlik, J.W., and Dietterich, T.G. (1990), Editors. *Readings in Machine Learning*. Morgan Kaufmann Publishers, Inc., Palo Alto, CA.

[SHG89] Shapiro, S.C., and the SNePS Implementation Group (1989). *SNePS-2 User's Manual*. State University of New York at Buffalo, Department of Computer Science.

[SHR87] Shapiro, S.C., and Rapaport, W.J. (1987). SNePS Considered as a Fully Intensional Propositional Semantic Network. In Cercone, N., and McCalla, G. (Eds.). *The Knowledge Frontier*. Springer-Verlag, NY, 263-315.

[STA86] Stanfill, C., and Waltz, D. (1986). Toward Memory-Based Reasoning. *Communications of the ACM*, Vol. 29, No. 12, 1213-1228.

[SUN92] Sun, R. (1992). A Connectionist Model for Commonsense Reasoning Incorporating Rules and Similarities. *Knowledge Acquisition*, 4, 293-321.

[SUN93] Sun, R. (1993). An Efficient Feature-Based Connectionist Inheritance Scheme. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 23, No. 2, 512-522.

[SZE93] Szepesvári, C., and Lörincz, A. (1993). Behavior of Adaptive Self-Organizing Autonomous Agent Working with Cues and Competing Concepts. Submitted.

[VAN94] Van Horn, K.S., and Martinez, T.R. (1994). The Minimum Feature Set Problem. Neural Networks, Vol. 7, No. 3, 494-494.

[WAT86] Waterman, D.A. (1986). *A Guide to Expert Systems*. Addison Wesley.

[WEB81] Webber, B.L., and Nilsson, N.J. (1981), Editors. *Readings in Artificial Intelligence*. Tioga Publishing Company, Palo Alto, CA.

[WER74] Werbos, P.J. (1974). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Ph.D. Thesis, Harvard University, Cambridge, MA.

[WET93] Wettschereck, D., and Dietterich, T.G. (1993). An Experimental Comparison of the Nearest-Neighbor and Nearest-Hyperrectangle Algorithms. To appear in *Machine Learning*.

[WOL94] Wollowski, M. (1994). Case-based Reasoning as a Means to Overcome the Frame Problem. In *Proceedings of the Seventh Florida AI Research Symposium (FLAIRS'94)*, 241-244.

[YOU93] Young, A., and Sattar, A. (1993). Nonmonotonic Inheritance Reasoning in Classifiers. In *Proceedings of the First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems (ANNES'93)*, 160-162.

[ZAR94] Zarndt, F. An Empirically Derived Test Suite for Machine Learning and Connectionist Algorithms. In preparation.

[ZLA90] Zlatareva, N. (1990). Considerations on Representing and Handling Human Common-Sense Knowledge. TASSO-Report, No. 10.