AN INCREMENTAL LEARNING MODEL FOR COMMONSENSE REASONING

Christophe Giraud-Carrier

Department of Computer Science
Brigham Young University
Provo, UT 84602

Tony Martinez

Department of Computer Science
Brigham Young University
Provo, UT 84602

ABSTRACT

A self-organizing incremental learning model that attempts to combine inductive learning with prior knowledge and default reasoning is described. The inductive learning scheme accounts for useful generalizations and dynamic priority allocation, and effectively supplements prior knowledge. New rules may be created and existing rules modified, thus allowing the system to evolve over time. By combining the extensional and intensional approaches to learning rules, the model remains self-adaptive, while not having to unnecessarily suffer from poor (or atypical) learning environments. By combining rule-based and similarity-based reasoning, the model effectively deals with many aspects of brittleness.

1. INTRODUCTION

Much effort has been devoted to understanding learning and reasoning in artificial intelligence [6, 19, 22]. 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 paper 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 [17]) 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 [20]. 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 [9, 20] 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 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 paper.

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 [3, 14, 16]. 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 [2].

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 commonsense. 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 [7] 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 [20]. 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.

The next section presents an overview of PDL2. Some of the more original aspects of the algorithm are then discussed. For more details, see [7].

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 pair p *covers* a pair q if and only if the set of examples represented by q is a proper subset of the set of examples represented by p. We say that p *subsumes* q if and only if p and q have the same output value and p covers q. A node's cover_count corresponds to the number of training pairs its stored pair subsumes. 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 [18]).

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 pty , counters, and/or cover_count,
- 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 [14]. 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 [20]. 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 [7], 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 [14], 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 [4, 8]. 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 [13]).

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 [20]. Insightful (and more general) discussions of commonsense, its implications, its nature and its manifestations can be found in [1, 5, 12].

Before discussing reasoning, we first briefly address the problem of representation in the attribute-value language. Reasoning tasks are often expressed in English 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 said 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 [17]. 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 subsumes p and p is an exception to at least one other rule. If p is an exception to another rule, say r', and r subsumes 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 [20]. 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 [10, 21, 23]. The combination of default (rule-based) reasoning and similarity-based reasoning in PDL2 provides a natural way (see [20], 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 [11], and [15] 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 [15]. 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 commonsense. 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 applic.

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 [11]. 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 is thus handled directly (B1), or extensionally.

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 other spaces (e.g., continuous)
- better characterization of conflicting defaults
- representation language (attribute-value language limited) and generalization
- source, form, generation and use of prior knowledge

ACKNOWLEDGMENTS

This work was supported in part by grants from Novell Inc., and WordPerfect Corp. We also wish to thank the three anonymous reviewers for many helpful comments.

REFERENCES

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

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

[3] Dietterich, T.G., and Michalski, R.S. 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, 1983, Chapter 3.

[4] Elman, J.L. 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, March 1991.

[5] Forguson, L. *Common Sense*. Routledge, 1989.

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

[7] Giraud-Carrier, C., and Martinez, T.R. 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*, 1993, 46-51.

[8] Giraud-Carrier, C., and Martinez, T.R. Seven Desirable Properties for Artificial Learning Systems. In *Proceedings of the Seventh Florida AI Research Symposium*, 1994, 16-20.

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

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

[11] Lifschitz, V. Benchmark Problems for Formal Nonmonotonic Reasoning. In *Proceedings of the Second International Workshop on Non-Monotonic Reasoning*, 1988, LNCS 346, 202-219.

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

[13] Martinez, T.R., Hughes, B., and Campbell, D.M. Priority ASOCS. To appear in *Journal of Artificial Neural Networks*, 1994.

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

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

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

[17] Reiter, R., and Griscuolo, G. On Interacting Defaults. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, 1981, 270-276.

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

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

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

[21] Sun, R. An Efficient Feature-Based Connectionist Inheritance Scheme. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 23, No. 2, March/April 1993, 512-522.

[22] Webber, B.L., and Nilsson, N.J. (Eds.). *Readings in Artificial Intelligence*. Tioga Publishing Company, Palo Alto, CA, 1981.

[23] Young, A., and Sattar, A. Nonmonotonic Inheritance Reasoning in Classifiers. In *Proceedings of the First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*, 1993, 160-162.

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 = F
Set creat_flag = T and modif_flag = F

```
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, 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 = F
      Case eq(n.p, q):
         If conc(n.p, q) then creat_flag = F 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 = F
      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 = F
            Else if num_assert(q) > 1 then modif_flag = T
   EndForAll              /* all nodes */
   For all generalizations g created in (*)
      If not ∃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 ∃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 of Section 4. We give them in terms of features and predicted output. All other attributes are don't-care.

zoo dataset:

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 a fish

lenses dataset:

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

voting and voting3 datasets:

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.