Machine Learning

Proceedings of the Seventh International Conference on Machine Learning

Edited by
Bruce W. Porter
and Ray J. Mooney

MACHINE LEARNING: PROCEEDINGS OF THE SEVENTH INTERNATIONAL CONFERENCE (1990)

University of Texas
Austin, Texas

June 21–23, 1990

EDITOR/WORKSHOP CHAIRS

Bruce Porter and Raymond Mooney, University of Texas, Austin

PROGRAM COMMITTEE

Ray Bareiss, *Vanderbilt University*
Gerald DeJong, *University of Illinois at Urbana-Champaign*
Ken DeJong, *George Mason University*
Brian Falkenhainer, *Xerox Palo Alto Research Center*
Douglas Fisher, *Vanderbilt University*
John Grefenstette, *Naval Research Laboratory*
Russ Greiner, *University of Toronto*
Kristian Hammond, *University of Chicago*
Rob Holte, *University of Ottawa*
Michael Kearns, *Massachusetts Institute of Technology*
John Laird, *University of Michigan*
Steven Minton, *NASA Ames Research Center*
Tom Mitchell, *Carnegie Mellon University*
Steven Muggleton, *The Turing Institute*
Michael Pazzani, *University of California at Irvine*
Lenny Pitt, *University of Illinois at Urbana-Champaign*
J. Ross Quinlan, *University of New South Wales*
Larry Rendell, *University of Illinois at Urbana-Champaign*
Jeffrey Schlimmer, *Carnegie Mellon University*
Alberto Segre, *Cornell University*
Jude Shavlik, *University of Wisconsin*
Devika Subramanian, *Cornell University*
Richard Sutton, *GTE Laboratories, Inc.*
Chris Watkins, *International Financial Markets Trading, Ltd.*

SPONSORS

Office of Naval Research, Artificial Intelligence and Robotics Program
Office of Naval Research, Cognitive Science Program
National Science Foundation, Knowledge Models and Cognitive Systems Program
Defense Advanced Research Projects Agency, Information Science and Technology Office
University of Texas Department of Computer Sciences

Editor *Michael B. Morgan*
Production Manager *Shirley Jowell*
Cover Designer *Jo Jackson*
Production and Composition *Technically Speaking Publications*

Library of Congress Cataloging-in-Publication Data

Machine learning: proceedings of the seventh international conference

(1990) / edited by Bruce Porter and Ray Mooney.
p. cm.
"This volume collects research results from ... the Seventh International Conference on Machine Learning, held June 21–23, 1990 at the University of Texas in Austin"–Pref.
Includes index.
ISBN 1-55860-141-4
1. Machine learning–Congresses. I. Porter, Bruce, 1956– .
II. Mooney, Ray, 1961– . III. International Conference on Machine Learning (7th : 1990 : University of Texas in Austin)
Q325.5.M34 1990
066.3'1--dc20

90-4763
CIP

MORGAN KAUFMANN PUBLISHERS, INC.
Editorial Office:
2929 Campus Drive
San Mateo, California
Order from:
P.O. Box 50490
Palo Alto, CA 94303-9953

Printed in the United States.

94 93 92 91 90 5 4 3 2 1

CONTENTS

Preface

Machine learning is a study of computational methods for acquiring knowledge and improving problem solving ability. Because of the breadth of this charter, machine learning includes a wide range of topics. This volume collects research results from twelve areas of machine learning which were represented at the Seventh International Conference on Machine Learning, held June 21–23, 1990 at the University of Texas in Austin.

The 165 technical papers submitted to the conference provide evidence that machine learning continues to mature and evolve. New areas of active research, such as robot learning, have emerged, presenting challenging new problems and applications. Furthermore, many papers described research that cuts across traditional boundaries in machine learning and synthesizes disparate results.

Tom Mitchell, Doug Lenat, Lenny Pitt, and Don Michie were invited to discuss their ongoing projects at the conference. Their sometimes unconventional views provided controversy and perspective.

The success of the conference is due to the concern and dedication of all those involved. In particular, the twenty-four members of the program committee earned the praise of the authors because of their comprehensive and insightful reviews. The conference staff—John Haradon, Yvonne Van Olphen, and Bess Sullivan—provided invaluable organizational skills. Finally, Shirley Jowell of Morgan Kaufmann Publishers produced this excellent proceedings.

The conference was funded in part by generous contributions from the Office of Naval Research, the Defense Advanced Research Projects Agency, and the National Science Foundation. Their long-term support for machine learning research has been critical to the success of this important field.

Bruce Porter
Ray Mooney
University of Texas, Austin

EMPIRICAL LEARNING

Knowledge Acquisition from Examples using Maximal Representation Learning

S Arunkumar and S Yegneshwar*
Department of Computer Science and Engg.
Indian Institute of Technology
Bombay - 400 076, India.
uunet!shakti!betaal!cse!sak
uunet!shakti!betaal!cse!yagi

Abstract

In this paper we describe a new Knowledge Acquisition technique based on the *learning from examples* paradigm. This technique uses the principle of *maximal representation* of attributes which clusters the examples into sub-descriptions using the frequencies, and orders the attributes based on how well it represents the class. The sub-descriptions aid in simple definitions of importance and redundancy of attributes for a class which are then used to discard unimportant attributes and thereby simplify class description. The resultant description is the simplest characteristic description which describes the class completely with respect to the learning examples and the attributes specified. This helps not only in describing the class well but also in potentially discriminating the current class from all other classes envisaged. An inference algorithm based on the frequencies of the sub-descriptions and importance of attributes is used to classify new examples. This system has been tested on two real-life applications, the results of which are highly encouraging.

1 Introduction

Knowledge acquisition is the process of acquiring knowledge to create expertise for solution of problems in a particular field of application. This is one of the main bottlenecks in the development of Expert Systems [Duda 83], and there have been many attempts to automate the process using learning techniques [Quin 86] [Mich 80] [Lee 86] [Paw 84] [Fu 85]. However, the emphasis of most of these systems is on learning discriminative description of the class (represented as decision trees [Quin 86][Lee 86][Craw 89] or as clusters [Mich 83]) which helps in performance, but does not generate an intelligible knowledge base. This is because the aim of discrimination is to give a description which deterministically separates examples of one class from examples of other classes, rather than to find a complete description of the given class based on the set of attributes specified. For instance, the two concepts chair and stool can be discriminated using the attribute backrest, though this attribute is not sufficient to describe either of them. Our emphasis here, is on learning the characteristic description of the class. There is evidence that humans too prefer characteristic description as highlighted in [Med 87].

A domain independent system which learns a characteristic description of the class and that determines the redundancy of attributes was proposed by the first author and first reported in [Doct 85]. The system described here is a modified version of the above work which in addition to having a modified learning algorithm and definition of redundancy, also learns importance of attributes for a class [Yeg 87] [Arun 89]. In this paper, we restrict to examples specified by binary valued attributes.

In the following section we explain the proposed learning of the class description from examples for binary valued attributes and prove its convergence. In section 3 we explain how redundancy of attributes for a class is determined. The importance of an attribute is defined in section 4. In section 5 we explain the process of inference based on category validity [Med 87]. We highlight the two applications in section 6, and conclude the paper in section 7.

2 Learning Class Description from Examples

The system learns descriptions of the classes for which examples have been given. Each learning example is represented as a vector or as (attribute,value) pairs. For instance an example of a stool may be represented as (number of legs, 4) (height, short) (shape of top, circular). The class description is represented as a binary tree called Generation Tree (GT).

2.1 The Generation Tree

The Generation Tree is learnt using the principle of *maximal representation* which states that at each node of the tree select that attribute whose most representative value has the highest cardinality. This learning criterion ensures that relatively more important at-

*This work is part of the Intelligent Systems Project at IIT, Bombay, India

tributes are selected earlier in the tree and the less important ones lower down. The Generation Tree is contructed top-down from the root to the leaves.

2.2 Algorithm for construction of GT

The GT is built using the criterion of *maximal representation* at each node.

1. Consider the given set of examples at the root node. Select the most representative attribute among the examples at the current node. This is done as follows:
   (a) If both the binary values occur for the given attribute among the set of examples at the current node, then find the cardinality of both the sets, where all the examples in the first set have one value (say 0) for this attribute and all the examples in the other set have the complementary value (value 1) for this attribute. Else, find the cardinality of the given set.
   (b) Find the maximum of the cardinality of the resulting sets (or set).
   (c) Repeat steps (a) and (b) for all the attributes and select that attribute as most representative which has the maximum cardinality as determined in step (b).
2. For both the values (0 and 1) branch off from the current node. At the left child node consider all those examples having value 0 for further branching, and at the right child consider all those examples having value 1 for further branching.
3. *Termination Condition :* If all the attributes have been selected at some node along all the paths of the tree or if the frequency along all the leaves is less than a specified threshold called *expansion threshold*, then stop. Else, go to step 2 and proceed in a depth first fashion.

Example 2.1 *Given the following set of examples specified by three attributes and represented in vector form, and the* expansion threshold *equal to 0.*

$$S = \{(101)(011)(101)(111)(101)(011)(111)\ (101)(111)(101)\}$$

We will show how the GT is built for this Example set. We have,

$$\{(\mid a_1 = 1 \mid = 8)\ (\mid a_1 = 0 \mid = 2)\ (\mid a_2 = 1 \mid = 5)\ (\mid a_2 = 0 \mid = 5)\ (\mid a_3 = 1 \mid = 10)\ (\mid a_3 = 0 \mid = 0)\}$$

where $\mid . \mid$ *stands for cardinality.*

Since a_3 *satisfies the* maximal representation learning *criterion, at the root node we select* a_3*. Then at the next node we select* a_1 *and proceeding similarly, we get the GT described in figure 1.*

Theorem 2.1 *The Generation Tree constructed by the algorithm described above converges in the limit.*

Proof : *By Glivenko-Cantelli Theorem,* $F_n(x) - F(x) \rightarrow 0$*, with probability 1 uniformly in* x*, where* F_n *is the empirical distribution function and* F *the true distribution. This implies that for a given* $\epsilon > 0$*,* $\exists n(\epsilon)$ *such that* $\mid F_n(x) - F(x) \mid < \epsilon$*, with probability 1,* $\forall n > n(\epsilon)$*.*

If we assume that the joint frequencies are such that in the "limiting description" there are no ties in the selection of attributes at all the nodes of the Generation Tree, we can choose ϵ *such that for all* $n > n(\epsilon)$*, chosen suitably, the structure of the Generation Tree stabilises. Once the structure stabilises the frequencies also stabilise. Furthermore, by the Central Limit Theorem, this convergence is at the rate of* $\sqrt{n}$*.*

2.3 Class Description from Generation Tree

The class description is represented as a binary tree where nodes are labelled by attributes and arcs by the corresponding attribute's value and joint frequency. The attribute at a node is selected using the *maximal representation learning* criterion explained in section 2.2. The Generation Tree is a disjunction of sub-descriptions. Each sub-description corresponds to a path in the GT, and is a conjunction of (attribute,value) pairs of that path. The sequence of the (attribute,value) pairs in the sub-description specifics that each such pair is conditioned on the previous pairs. It is this sequence which helps determine the importance of attributes. In Example 2.1 there are three sub-descriptions, viz. $\{(a_3 = 1)\ \&\ (a_1 = 1)\ \&\ (a_2 = 1)\ 0.3\}$, $\{(a_3 = 1)\ \&\ (a_1 = 1)\ \&\ (a_2 = 0)\ 0.5\}$, $\{(a_3 = 1)\ \&\ (a_1 = 0)\ \&\ (a_2 = 1)\ 0.2\}$.

The binary tree structure for class description is found to be powerful enough for practical applications. It also helps in evaluation of importance of attributes for a class, and aids in determining redundancy of attributes in a simple manner.

3 Redundancy of an attribute at a node of a GT

The aim of finding if an attribute is redundant at a node is to simplify the class description by pruning the Generation Tree. It may also lead to finding redundant attributes for a class. A simple class description helps in easier explanation, and the collection of less information from the user. The latter could mean saving in terms of time and money for the user if the attribute is a complex test in a diagnostic context.

In the GT framework, it is very easy to determine redundancy of an attribute at a node. The GT is constructed top-down, whereas redundancy is determined bottom-up. The reason for this is as follows : Only at the bottom-most node of a path, redundancy can be determined at any time because, at any other node the correlation of subsequent attributes is not known. Since the attribute at the bottom-most node is conditioned on all the earlier attributes no more information is required and the redundancy check can be applied at this node.

Definition 3.1 *An attribute is said to be redundant at a node if the node is the lowest in the path and both the domain values occur with equal frequency.*

Note: The equality is in the ideal case. In practice, however, if the percentage difference in frequency is less than a specified threshold then, that node can be deleted.

Definition 3.2 *An attribute is redundant for a class if it is redundant at all the nodes where it is chosen, i.e., it does not occur at any node of the GT after the redundancy check is applied.*

Definition 3.3 *A class is said to be a* null class *if each element of the cartesian product of the domain of attributes occurs with equal frequency. In the GT framework such a class should be represented by a single node.*

Note: The definition of redundancy helps reduce the GT in such a case to a single node. This is illustrated by the following example.

Example 3.1 *In this example we show that the given set of samples S={(1 1) (1 0) (0 1) (0 0)} represents the* null class. *This also illustrates redundancy at a node and attribute redundancy for a class. The reduction of the initial GT to a single node is described pictorially in figure 2.*

Note: The redundancy of an attribute for decision tree and clustering systems is based on the discriminative power of the attribute whereas in KAHLE, an attribute is redundant because it does not add to the description of the class.

4 Importance of an attribute

The Importance of an attribute is the relevance of the attribute for a particular class. It is denoted as $I(a_r \mid C_x)$, read as Importance of an attribute a_r given class C_x. The importance of attributes for a class not only aids in inference under uncertainty but also helps prune the GT by enabling dropping of attributes.

Since the Importance of an attribute is judged by the representativeness of the attribute in the class, we consider the *Importance of an attribute given the class as directly proportional to the frequency at the node at which it occurs, and inversely proportional to the distance of the node from the root.*

The importance lies between 0 and 1, and satisfies the following property:

$$\sum_{r=1}^{n} I(a_r \mid C_x) = 1 \qquad (1)$$

As a first approximation we have chosen the following form for $I(a_r \mid C_x)$ which satisfies the above constraints.

$$I(a_r \mid C_x) = \sum_{p=1}^{t_r} (2/(n*(n+1))*(n-d_{rp})*f_{rp} \quad (2)$$

where
n is the distance from leaf to the root node (i.e. equal to the number of attributes considered),
2/(n*(n+1)) is the normalising factor,
t_r is the number of occurrences of attribute a_r in the GT,
d_{rp} is the distance of the pth occurrence of attribute a_r from the root, and
f_{rp} is the joint frequency at this node.

Note: In both decision tree and clustering systems the importance of an attribute for a class is not explicitly evaluated. However, in decision tree systems it is implicit that an attribute occuring higher in the tree is more important for the application (i.e., for all the classes) than those occuring lower down, because the entropy measure is used to select the atribute at each node.

5 The Inference Process

The inference process used in our system is a modified version of Category Validity [Med 87] which is defined as the probability of the attribute value set given the class. The attribute value set corresponds to a path in the GT. The modification was necessary to help classify test examples having missing attribute values. The process is described below:

1. Set the first class C_1 to C_i.
2. Start with the root node of the GT corresponding to the class C_i.
3. At each node of the GT traverse that arc whose attribute value is the same as that of the input example e. If the value is different from all the arcs existing at that node then, output validity as zero and stop. If the value is missing then, traverse all the arcs at that node. Repeat this for all the nodes. Then, evaluate $f(e \mid C_i)$ the Validity of class C_i as shown below:

$$\begin{aligned} f(e \mid C_i) &= \{f(e_{p1} \mid C_i) * \sum_{l=1}^{n1} I(a_l \mid C_i) + \\ &\quad f(e_{p2} \mid C_i) * \sum_{l=1}^{n2} I(a_l \mid C_i) + ... + \\ &\quad f(e_{pk} \mid C_i) * \sum_{l=1}^{nk} I(a_l \mid C_i)\} \\ &= \sum_{j=1}^{k} \{f(e_{pj} \mid C_i) * \sum_{l=1}^{nj} I(a_l \mid C_i)\} \end{aligned}$$

where
$f(e \mid C_i)$ is the Validity of class C_i given example e,
pj is the jth path which the evidence matches either completely or incompletely (incompletely if the example has missing attribute values),
e_{pj} is the set of (attribute,value) pairs along path pj,
$f(e_{pj} \mid C_i)$ the validity of class C_i given e_{pj}, is the leaf frequency along path pj, and
nj is the number of attributes along path pj for which value is available in e.

4. If C_i is the last class then go to next step else repeat steps 2 and 3 for the next class, viz., for $C_i := C_{i+1}$.
5. Select that C_i for which $f(e \mid C_i)$ is the maximum. If the maximum value is equal to zero then output "Not Classified" and exit; else output C_i.

Example 5.1 *Suppose we have three classes for which the GTs are as shown in fig. 3.*

Suppose two examples to be classified are $S_1 = \{(a_1 = 1)\,(a_2 = 0)\,(a_3 = ?)\}$ *and* $S_2 = \{(a_1 = 1)\,(a_2 = 1)\,(a_3 = 1)\}$, *where ? stands for missing value then,*

$$\begin{aligned} f(e_{p1} \mid C_1) &= f((a_1 = 1)(a_2 = 0)(a_3 = 1) \mid C_1) \\ &= 0.6 \\ f(e_{p2} \mid C_1) &= f((a_1 = 1)(a_2 = 0)(a_3 = 0) \mid C_1) \\ &= 0.3 \\ I(a_1 \mid C_1) &= 2/(3 * (3 + 1)) * (3 - 0) * 1 = 0.5 \\ I(a_2 \mid C_1) &= 2/(3 * (3 + 1)) * ((3 - 1) * 0.9 + \\ &\quad ((3 - 1) * 0.1) = 0.33 \end{aligned}$$

Similarly the other importances can be calculated. The values are as follows:

$$\begin{aligned} I(a_1 \mid C_1) &= I(a_1 \mid C_2) = I(a_1 \mid C_3) = 0.5 \\ I(a_2 \mid C_1) &= I(a_2 \mid C_2) = I(a_2 \mid C_3) = 0.33 \\ I(a_3 \mid C_1) &= I(a_3 \mid C_2) = I(a_3 \mid C_3) = 0.17 \\ f(S_1 \mid C_1) &= f(e_{p1} \mid C_1) * (I(a_1 \mid C_1) \\ &\quad + I(a_2 \mid C_1)) + f(e_{p2} \mid C_1) * \\ &\quad (I(a_1 \mid C_1) + I(a_2 \mid C_1)) \\ &= 0.6 * (0.5 + 0.33) + 0.3 * (0.5 + 0.33) \\ &= 0.75 \\ f(S_1 \mid C_2) &= f(e_{p1} \mid C_2) * (I(a_1 \mid C_2) \\ &\quad + I(a_2 \mid C_2)) + f(e_{p2} \mid C_2) * \\ &\quad (I(a_1 \mid C_2) + I(a_2 \mid C_2)) \\ &= 0.1 * (0.5 + 0.33) + 0.7 * (0.5 + 0.33) \\ &= 0.67 \\ f(S_1 \mid C_3) &= 0.0. \text{ [since } S_1 \text{ does not match any} \\ &\quad \text{path of } C_3\text{]}. \end{aligned}$$

Therefore, S_1 is classified as C_1. S_2 cannot be classified since it does not match (even incompletely) any path of any of the three GTs.

6 Practical Applications

The system has been implemented as a set of programs in Pascal on an MC68000 Unix machine. The inputs are learning examples, *expansion threshold*, sample size of each class, attribute names, and the testing examples. For each class, the frequency of "rightly classified", the frequency of "not classified" the frequency of "wrongly classified" examples, and the number of sub-descriptions are output.

Medicine has the best potential for application of learning systems as there is tremendous variation in the example cases of each disease. Here, a class is a disease and an example is a patient record. We have chosen a particular problem of stomach diseases concerning classification of hiatal hernia and gallstones as the first experiment. The second experiment of characterisation of the two political parties of USA, viz. Democrat and Republican helps substantiate our model.

Application 1[1]*:* This is a two class example involving either hiatal hernia or gallstones. Examples are specified by 11 binary valued attributes. The total number of examples are 107 of which in the first experiment 54 was used for learning and 53 for testing. In the second experiment these example sets were interchanged. The *expansion threshold* in both the experiments is 0.20. The performance details of the two experiments are given in tables 1 and 2 respectively.

The *classification accuracy* defined as (freq. of rightly classified / freq. of totally classified) is approx. 80% for the first experiment and 70% for the second. These are quite good considering that the classes are fuzzy and are overlapping. The number of sub-descriptions learnt are much lower than the number of distinct examples as seen from tables 1 and 2. In both the experiments, attributes a_4, a_7 and a_8 have an importance value less than 0.04 (which is very low) for both the classes. Hence these attributes were deleted and the experiment was carried out for both the databases. The number of sub-descriptions for both the classes are lower than that of the earlier experiment, and substantially lower than the number of distinct examples. *The Generation Tree learnt is thus compact.* The performance details given in table 3 is comparable to the performance of the earlier experiment.

The performance after the deletion of the attributes a_4, a_7, and a_8 (since these attributes once again had very low importance value) for the second database is as shown in table 4. The number of sub-descriptions is once again substantially lower than the number of distinct examples. *In this case the accuracy has improved from 70% to 79% which substantiates the fact that having more than the optimal number of attributes tends to degrade the performance [Chan 75].*

Application 2 : The database is the 1984 Congressional voting pattern records consisting of two classes, viz. Democrats and Republicans. There are sixteen binary valued attributes for which each member has to vote. The total number of examples is 435 of which 232 has all the attribute values and 203 has at least one missing attribute value. So, the first set is used for learning and the second set for testing. The value of *expansion threshold* is 0.25. The results of the study are as described in table 5.

The classification accuracy is equal to 94.5%. This compares favourably with [Schl 87] where the accuracy reported is between 90% to 95%.

Since the importance of attributes a_2 and a_{10} were zero for both the classes, and the importance of attributes a_1, a_{11} and a_{13} were very low they were

[1]This data has been taken from "Feature selection for binary data - medical diagnosis with fuzzy sets" by J.C.Bezdek in proc. of AFIPS, 1976, pp.1059-1062.

dropped and the system was tested with the same *expansion threshold*. The number of sub-descriptions for both the classes after deletion of attributes is lower than for the above experiment and much lower than the sample sizes of the two classes. The frequency details of the experiment given in table 6 is exactly the same as in the previous experiment.

7 Conclusions

We have described a new domain independent learning model for knowledge acquisition from examples specified by binary valued attributes using the *maximal representation learning* criterion. The main features of this model are learning of a provably convergent characteristic description of a class, and learning of importance and redundancy of attributes. The characteristic description generated here is the simplest definition having all the information of the learning examples. This description can potentially discriminate the class from all other classes envisaged. The concept of importance and redundancy of attributes helps in learning a compact class description without degradation in performance.

The study demonstrates that our approach learns a compact characteristic description of the class without sacrificing performance for two diverse applications and is hence encouraging. In the second experiment though all the test examples had at least one missing attribute value, the performance is good.

We have restricted to binary valued attributes in this paper. However, this is not a limitation of the method. The learning technique discussed in this paper can be used for non-binary (nominal and real) valued attributes with a m-way branching at each node instead of a two-way branching. All the definitions for binary valued attributes are easily extendable for this case. The implementation of the system for these attributes is complete, and empirical validation is in progress.

Acknowledgements We thank David Aha of the University of California, Irvine for providing us the 1984 US Congressional voting records database.

References

[Arun 89] Arunkumar S. and Yegneshwar S., "An Experimental Study of a new approach to Knowledge Acquisition from Examples", Tech. Rep. No.TR-12-89, Dept. of Computer Science And Engineering, Indian Institute of Technology, Bombay, India, Dec 1989.

[Chan 75] Chandrashekaran B. and Jain A.K., "Independence, Measurement Complexity, and Classification Performance", I.E.E.E. Transactions on Systems Man and Cybernetics, vol. SMC-5, 1975, pp.240-244.

[Craw 89] Crawford S.L., "Extensions to the CART algorithm", Intl. Jnl. of Man-Machine Studies, vol.31, 1989, pp.197-217.

[Doct 85] Doctor M., "Knowledge Acquisition for Expert Systems" BTech. Dissertation, Dept. of Computer Science and Engineering, Indian Institute of Technology, Bombay, India, 1985.

[Duda 83] Duda R.O. and Shortliffe E.H., "Expert Systems Research", Science, vol.220, 1983, pp.261-268.

[Fu 85] Fu Li-Min and Buchanan B.G., "Inductive Knowledge Acquisition for Rule based Expert Systems", Technical Report No. KSL-85-42, Knowledge Systems Laboratory, Dept. of Computer Science, Stanford University, USA, 1985.

[Lee 86] Lee W.D. and Ray S.R., "Probabilistic Rule Generator:A new methodology of variable-valued logic synthesis", Proc. of I.E.E.E. Conference on Multiple-valued logic, 1986, pp.164-171.

[Med 87] Medin D.L., Wattenmaker D.W., and Michalski R.S.,"Constraints and Preferences in Inductive Learning : An Experimental Study of Human and Machine Performances", Cognitive Science, vol.11, 1987, pp.299-339.

[Mich 80] Michalski R.S. and Chilauski R.L.,"Knowledge Acquisition by encoding Expert Rules vs. Induction from Examples : A case study involving soybean pathology", Intl. Jnl. of Man-Machine Studies, vol.12, 1980, pp.63-87.

[Mich 83] Michalski R.S. and Stepp R.E., "Learning from Observation : Conceptual Clustering" in R.S.Michalski, J.G.Carbonell, and T.M.Mitchell (Eds.), Machine Learning : an Artificial Intelligence approach, Tioga, Palo Alto, CA, USA, 1983.

[Paw 84] Pawlak Z., "Rough Classification", Intl. Jnl. of Man-Machine Studies, vol.20, 1984, pp.469-483.

[Qu'n 86] Quinlan J.R., "Induction of Decision Trees", Machine Learning, vol.1, num.1, 1986, pp.81-106.

[Scl. /] Schlimmer J.C., "Concept acquisition through representational adjustment", Doctoral dissertation, Dept. of Information and Computer Science, University of California, Irvine, U.S.A., 1987.

[Yeg 87] Yegneshwar S., "Domain Knowledge Acquisition using Maximum Invariance as a Learning Criterion for binary valued attributes", Ph.D. seminar report, Dept. of Computer Science and Engineering, Indian Institute of Technology, Bombay, India, 1987.

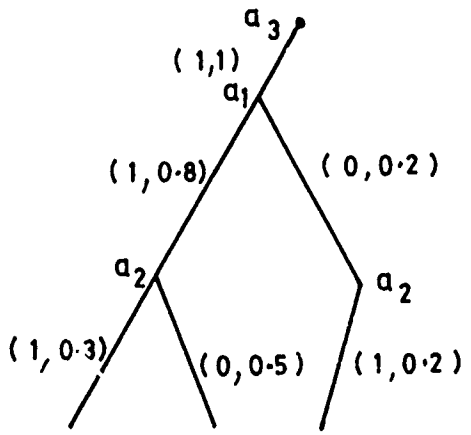


Figure.1. The Generation Tree

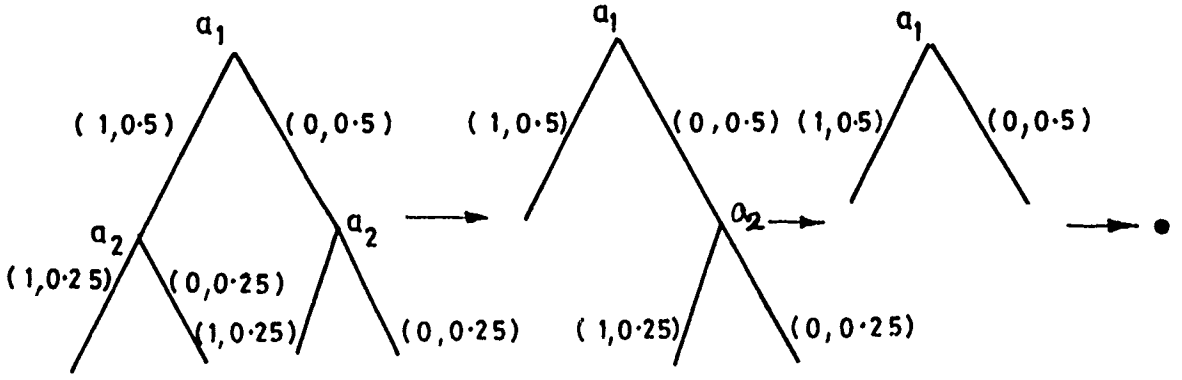


Figure.2. GTs Depicting Redundancy And Null Class

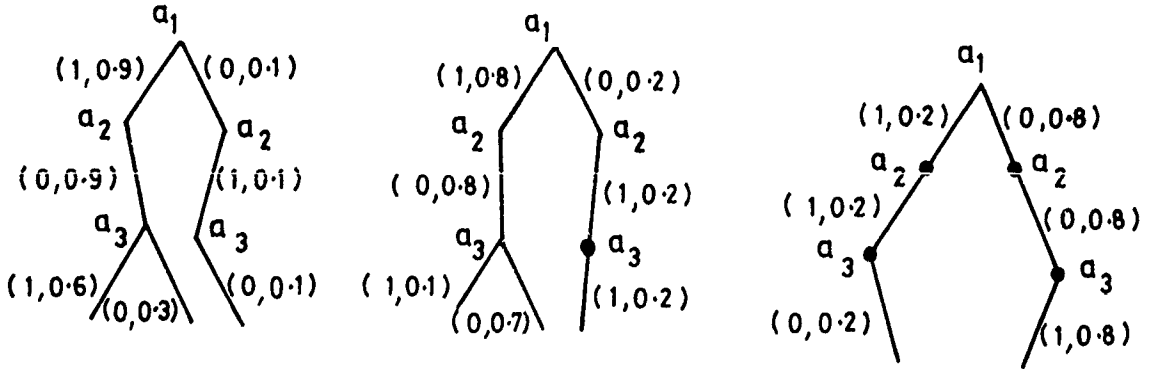


Figure.3. GTs For The Three Classes

Table 1: *Frequency details for the first database*

Class	freq. of learning examples	number of sub-des	freq. of rightly classd.	freq. of not classd.	freq. of wrongly classd.
1	29	11	26	0	2
2	25	11	16	0	9

Table 2: *Frequency details for the second database*

Class	freq. of learning examples	number of sub-des	freq. of rightly classd.	freq. of not classd.	freq. of wrongly classd.
1	28	9	22	1	6
2	25	9	15	1	9

Table 3: *Frequency details after deletion of attributes for first database*

Class	freq. of learning examples	number of sub-des	freq. of rightly classd.	freq. of not classd.	freq. of wrongly classd.
1	29	5	26	1	2
2	25	7	13	1	10

Table 4: *Frequency details after deletion of attributes for second database*

Class	freq. of learning examples	number of sub-des	freq. of rightly classd.	freq. of not classd.	freq. of wrongly classd.
1	28	5	26	1	2
2	25	7	14	1	10

Table 5: *Frequency details for the second experiment*

Class	freq. of learning examples	number of sub-des	freq. of rightly classd.	freq. of not classd.	freq. of wrongly classd.
1	108	15	57	1	2
2	124	14	134	0	9

Table 6: *Frequency details after attributes are deleted*

Class	freq. of learning examples	number of sub-des	freq. of rightly classd.	freq. of not classd.	freq. of wrongly classd.
1	108	11	57	1	2
2	124	12	134	0	9

KBG : A Knowledge Based Generalizer

Gilles Bisson
Laboratoire de Recherche en Informatique
Université Paris-sud, bât 490
91405 Orsay Cedex France
email : bisson@lri.lri.fr, phone : (33) 69-41-63-00

Abstract*

Fundamentally, generalization can be seen as a technique for performing compression of information ; the result of this process provides the *basis* to learn new knowledge. However, in all domains, the use of an elementary process requires the implementation of both fast and efficient algorithms. In this paper, we present a new generalization mechanism, inspired by the structural matching algorithm. The principle of our method consists in using the domain theory in order to perform the saturation of the examples given by an expert. We will see that this "rough" approach provides some advantages both for generalization quality and for construction rapidity.

1 Introduction

At the present time, a great number of generalization systems have been written and work. So, we can question the neccessity of adding a new one to this list. In fact, KBG corresponds to an extension of some existing and running systems such as AGAPE [Bollinger 86] and OGUST [Vrain 90], both based on the Structural Matching algorithm.

The system KBG is a generalizing tool belonging to the family of "Inductive Learning" and more accurately to the one of "Constructive Learning" [Kodratoff, Michalski 90]. It can learn a recognition function of a concept from an example set which illustrates this concept. The system has been developed in the frame of the PERSPICACE project [Cannat 88] whose aim was to apply techniques of Machine Learning to the domain of Air Traffic Control (ATC) in order to build an Expert Knowledge Base in Radar Conflict Resolution. Previous work, directed by the CENA ("Centre d'Etudes de la Navigation Aérienne", Athis-Mons, France), has pointed out the difficulties in modeling the tasks of air traffic controllers.

In this real world domains there were several problems to solve such as : the presence of both *symbolic* and *numeric* descriptors ; the necessity to express the data in *first order logic* because each entity can exist with a various number of occurrences ; the presence of complex *background knowledge* containing some *calculations* between the descriptors. Lastly, the great size of the learning set (examples and domain theory) required that any system *rapidly* perform generalization. In order to implement a system able to resolve simultaneously all these constraints, we needed to greatly modify the existing algorithms and we developed a new generalization method mainly based on the previous *saturation* of examples.

This paper is organized in two parts. In the first one, we detail our motivations and the advantages of our approach. In the second one, we present and explain more accurately the running of the system.

We must emphasize that KBG is a generalization tool but not a discrimination one : indeed, the aim of this system is to obtain *efficiently* a most *specific* generalization from an example set (without counter-examples), but not to build automatically a set of discrimination rules between the different concepts. Thereby, KBG is currently "just" a tool allowing the expert (or another part of the learning system) to build up more easily the rules modeling the application domain.

2 Structural Matching

2.1 Definitions

We will begin by briefly recalling the definitions of generalization and of Structural Matching [Kodratoff 86] as used in both AGAPE and OGUST. The main goal of the structural matching method is to limit the information loss during the generalization process by minimizing the use of the dropping rule [Michalski 83].

* This work is partially supported by CEC through the ESPRIT-2 contract MLT 2154 and also by MRT through PRC-IA.

Def1 : We say that an expression G is a generalization formula, modulo a domain theory T, of the examples set $E_1, E_2, ..., E_n$, if there exists N substitutions σ_i such that for each i : $\sigma_i(G)$ is included into E_i, modulo the domain theory T.

Def2 : We say that $E_1, E_2, ... E_n$ structurally match if there exists a formula F and N substitutions σ_i such that for each i : $\sigma_i(F)=E_i$. The formula F constitutes a correct generalization of the examples. To put the examples into Structural Matching aims at finding F.

In order to put the examples into Structural Matching [Ganascia 87], [Kodratoff 88], we first use the theorems and taxonomies given in the background knowledge as well as the commutativity, associativity and idempotency properties of conjunction. The elementary process of the Structural Matching algorithm is split into two successive steps which are repeated until there are no more constants within the examples :

1) The system chooses a constant O_i in each example and turns it into a generalization variable GV. The chosen constants must be as "similar" as possible; these choices are driven by some heuristics. During the introduction of GV, the corresponding instantiations are memorized into some links associated with each example.

2) Then, the terms containing the generalization variable GV are matched, taking care not to modify the previously introduced variables.

In the second stage, in order to satisfy the Structural Matching definition, we apply the dropping rule to the parts of the examples which have not been treated by the previous algorithm. Finally, the generalization formula is easily obtained by computing the intersection between the links associated with each example and by keeping the common terms of the examples.

2.2 Algorithm Analysis

In the previous algorithm, informations given by the domain theory are used according to the needs. Thereby, at a given time, the set of facts handled is relatively small. So, this method is theoretically usable with any size of examples and domain theory. However, in practice we realize that we often use the knowledge base and that therefore some pieces of information are infered twice or more. Indeed, the inferences are first performed, when the system is searching for the most similar constants, and secondly, when it puts effectively the terms in Structural Matching. All these computations slow down the generalization process. If this problem can be solved by memorizing the deduced instances, there is a more embarrassing one about the choice of the theorems in the background knowledge to apply. As a matter of fact, for lack of meta-knowledge, the choice of suitable rules is mainly driven by syntactical criteria. As we can see in the following example, this solution is not totally satisfying :

Background knowledge :

T1 : IF rect (X), blue (X,Y) THEN flag (X)
T2 : IF rect (X), red (X,Y) THEN flag (X)
T3 : IF square (X), blue (X,Y) THEN signal (X)
T4 : IF square (X), red (X,Y) THEN signal (X)
T5 : IF square (X) THEN shape (X)
T6 : IF rect (X) THEN shape (X)

Examples :

e1 : rect (a), red (a, hot), square (b), blue (b, light);
e2 : square(c), red (c, dark), rect (d), blue (d, roy);

By using simple syntactical criteria to choose the rules, such as the relative complexity of the theorems fired (number of premises and variables), we are going to match the constants (a, c) and (b, d) because it is simpler to match the colors (red, blue) than the forms (rect, square). So, we obtain the following generalization :

G = shape (X), shape (Y), red (X, Z), blue (Y,T)

But, a more detailed examination of the examples and of the domain theory suggests that the crossed matching (a,d) and (c,b) is much better in terms of information content. In this way, we obtain the generalization :

G' = rect (X), square (Y), red (Z, W1), blue (T, W2), flag (X), signal (Y)

2.2 Why The Saturation Approach ?

Here, our argument is that the syntactical criteria can lead to some misinterpretations of the semantics of the domain. Therefore, for lack of meta-knowledge which would allow to elaborate a coherent strategy for the choice of rules, we think that ultimately the best approach will be to saturate initially the examples with the domain theory provided by the expert. This solution may seem very questionable, so we are going to discuss some further advantages :

There are three kinds of advantages. Firstly, the deduction step is done only once, when the system saturates the examples ; thereby, the theorem prover used can be both simple and efficient because it just consists in a forward chaining engine. Secondly, with this approach, when the system is searching for the "most similar constants" it already knows all the properties

verified by the constants ; thus, the constants are chosen with full knowledge of the facts and the generalization will be more relevant. Last but not least, as the deductions are done once and for all, and no longer for each matching step, the learning will be faster in spite of the data increase.

Of course, the main drawback of this approach is the possibility of running into combinatorial explosion. However, several limiting factors exist. On one hand, the facts are handled at the example level : therefore, if there are E examples and N facts (terms) on average, we must execute E propagations of N facts instead of one propagation of ExN facts. On the other hand, the example descriptions are usually composed by less than a hundred facts; moreover, as part of similarity based learning the domain theories are relatively small. Note that the use of recursive rules in the background knowledge is not a problem as long as they do not generate an infinite list of instances ...

Practically, during the project PERSPICACE, we have used without problem a domain theory containing 250 complex rules (with recursions) and 20 examples of 50 terms each [Cannat 88].

3 The Algorithm

3.1 Introduction

The generalization process is performed by the system in two steps. Firstly, using the background knowledge given by the user, the system completes the examples with the help of a forward chaining engine (saturation). Secondly, the learning process begins, working on the fact bases, one for each example, built with the help of the inference engine. The generalization formula will be composed of only the predicates which have at least one instance in each fact base, expression of the saturated examples. This set of predicates is named GP. The generalization is performed predicate by predicate . For each predicate, we gather the "most similar" instances together, in order to turn them into variables. Roughly, the algorithm is the following :

```
* For each predicate P belonging to the GP set
  * Until there remains one ungeneralized instance
      - Match the most similar instances of P.
      - Name the generalization variables Vi and
        create the links between them.
      - Append the new predicate P(V1, ...) to the
        generalization formula.
```

Firstly, we have to define the similarity computation between the instances of a given predicate. In order to execute this computation as fast as possible, we need to index the information used ; to achieve that, we build for each of the constants, composing the examples, a list gathering the typically verified properties. This list will be call the "signature of the constant".

3.2 Signature Between Constants

The "signature of the constant", noted SIG, is the basis of the similarity computation. It expresses an exhaustive abstract of the properties verified by the constant in the fact bases. Since we saturate the examples at the beginning of the learning session, the signature need only be computed once, the end of the saturation step. In KBG, we can split the constants into two types that we call "objects" and "values" (notice that the previous algorithms, did not handle the values). The objects are constants without semantics : in other words, their significance is only provided by the predicates in which they occur. The values are typed constants and contain their own meaning. Here is an example :

Let the type "size" which handles the values :
(small, medium, large, giga)
Let the expression :
E = square(a) & outsize(a, small) & weight(a, 50)

According to our previous definition, the constant "a" is an object and the constants "small" and "50" are values. We can easily see the different semantics associated with these two kind of constants : the objects express links between the properties (predicates) of the same entity, whereas the values quantify those properties. Consequently, the signatures of objects and values are different. In the case of an object, its signature will be composed by the list of the occurrences (predicates, positions) of this constant within the example :

E = square(a) & outsize(a, small) & weight(a, 50)
SIG (a) = (square1 outsize1 weight1)

The values are completely local to the predicates in which they occur, thereby the signature of a value is itself. For instance : SIG(50)=50.

* Redundant instances :

However, the completion process can sometimes have some drawbacks ! Indeed, the same information can be deduced several times. Consider this problem :

E1 = color (A) & square (A)
E2 = color (B) & square (C)
We also know the two rules :
R1 : square (x) => rectangle(x)
R2 : rectangle(x) => shape(x)
After the completion of the examples, we obtain the following signatures :
SIG(A) = (color square rectangle shape)
SIG(B) = (color)
SIG(C) = (square rectangle shape)

In this case, for the examples E1 and E2, the occurrences "rectangle" and "shape" are not relevant because they provide no information with respect to "square". This is an annoying problem for two reasons : firstly, the signature comparison will be perverted and consequently the generalization; secondly, we will have an unnecessarily large number of instances to treat. Thereby, we need to delete this kind of information when the completion step is achieved. This operation will be easily performed with the following algorithm, which complexity is linear in accordance to the number of instances in the facts' base :

* For each predicate P owning at least one instance :
  If there is in the background knowledge a theorem T whose premises are made with the predicate list (Q1...Qn) named LP, such that :
  1) All the instances of P have been deduced by the theorem T (at least)
  2) Let LPI the predicate list of LP whose arguments are identical to those of P. Instances of P must be identical to those of one of the predicates of LPI (<=> same number of instances).

  Then the instances of P are not relevant, thus :
  * If GP owns P then P is not generalized.
  * P does not appear in the constant signature.

3.3 Similarity Between Constants

In first order logic, the examples can be represented as graphs where the nodes are the constants of the examples and the edges the predicates (or properties) linking these constants. So, in this representation, the generalization problem becomes one of finding the "greatest" subgraph common to all the examples : unfortunately, this is a NP problem ! Thus, to match correctly and rapidly these constants, we consider that the generalization process consists of the gathering, under the same variable name, of the most similar constants. In order to achieve this, we are going to define a notion of similarity (distance) between constants. This measure is noted SIM and its computation depends on the constant type.

* Similarity between objects :

In the case of objects, we just look at the common properties between the two signatures, for instance, by measuring the length of the lists' intersection. However, It is necessary to balance this measure with the number of differences observed. Otherwise the constants owning a lot of properties will be match together. Practically, we use the following formula :

$$SIM(X,Y) = \frac{LENGTH(SIG(X) \cap SIG(Y))}{LENGTH(SIG(X) \cup SIG(Y))}$$

We must emphasize that by sorting the signature list before computing the similarity, the complexity of this operation becomes linear in relation to the number of items. Here is a computation example :

E1 = square(a) & red(a) & small(a)
E2 = rect(b) & red(b) & square(c) & green(c) & small(c)
SIG(a) = (square small red);
SIG(b) = (rect red);
SIG(c) = (square small green)
SIM(a,b) = 1/3 ; SIM(a,c) = 2/3

* Similarity between values :

In the case of values, the similarity measure is computed between the values owned by the same predicate, according to the constant type. For instance, if the constants belong to an ordered type, we can classically define the measure of similarity as :

Let the type "size" : (small, medium, large, giga)

$$SIM(a,b) = \frac{(Interval\text{-}length - Place\text{-}difference(a,b))}{Interval\_length}$$

SIM (small, medium) = 3/4 = 0,75
SIM (medium, giga) = 2/4 = 0,50

3.4 Similarity Between Instances

Practically, the predicates are n-tuplets and the arguments composing their instances are objects and values. Therefore, for each argument, the computation of the similarity will be performed according to the constant type. For a N arity predicate, the distance between two instances I1 and I2 will be defined as the average of the similarity between each arguments $A1_n$ and $A2_n$.

3.5 Matching Between Instances

The matching of the most similar instances of a predicate P is the basic operation which allows to create a new term in the generalization. It consists of selecting for each example E_i, one instance I_i of P in the fact base associed to E_i, taking care to maintain as far as possible the greatest similarity between the items of the resulting vector (I_1 ... I_E). However, for a given predicate, we can not compute the optimum matching; indeed, if we have E examples and an average of N instances for each one, the complexity of this computation is exponential in $O(N^E)$.

So, we use the following heuristic to obtain a reasonable complexity. The elementary step of the process is to select in the fact base associated with an example one instance (not yet generalized) I ; then, for each remaining example we choose the most similar instance (generalized or not). Finally, we obtain a generalizable instance set. The computation is finished when all of the instances are treated. The great advantage of this method is firstly, its simplicity and secondly, a

reasonable complexity in $o(N^2.E)$. Another advantage of this heuristics is that we can easily control the application of idempotence (I => I&I), by modifying the stopping test of the previous algorithm ; if we do not need idempotency, the system will match only the instances not yet generalized. The main drawback is that the quality of the matching depends a lot on the choice of the first instance ; thereby, this heuristic can be very noise sensitive !

3.6 Links Between Variables and Constants

The instance matching process provides us with a vector V : $(I_1 \ldots I_E)$, in which each instance is composed of a list of N constants, N representing the arity of the current predicate. This vector can be seen as a matrix of constants $M_{N,E}$; in this way, creating a variable corresponds to grouping, under the same symbol name, one column of this matrix. For a given data type, the different variables built during the generalization process are not totally independent because their definitions often have some common parts (list of substituted constants). Therefore, independently of the variable type (object or value) we must define some links between the variables, specifying the coherence constraints :

* (X = Y) : both variables are always instanciated with the same items.
* (X ≠ Y) : both variables are never instanciated with the same items.
* (X ~ Y) : the variables can be either the same or different (May Be the Same).

Traditionally, the MBS links are not explicitly given in the generalization : as a matter of fact, in logic when two variables have the same name, they can be either equal or different. On the other hand, when we compute a generalization, we notice that the MBS links are less numerous than the difference ones. Therefore, in order to increase the readability of the result, we have chosen another convention in our system : it is the difference links which are implicit.

* Exclusive links :

The MBS links are not always simultaneously true. For example, we can write the links (X~Y) and (Y~Z) without having, in the examples, the equality X=Y=Z ; in this case, we have lost some information. Therefore, the relation between (X~Y) and (Y~Z) must be expressed as a logic NAND rather than a simple AND. We illustrate this point in the following example :

E1 = rect (a) & red (a) & rect (b) & large (b)
E2 = rect (c) & red (c) & rect (d) & red (d) & large (d)
E3 = rect (e) & red (e) & rect (f) & large (f)

$$G = rect(X) \,\&\, rect(Z) \,\&\, red(X) \,\&\, red(Y) \,\&\, large(Z) \,\&\, (X \sim Y) \,\&\, (Y \sim Z)$$
$$with\ X = (a\ c\ e),\ Y = (a\ d\ e),\ Z = (b\ d\ f)$$

We need to be able to detect this kind of information. Saying that two links L1 and L2 are verified at the same time means that the intersection between the example sets where L1 is true and where L2 is true, is *empty*. Then, the idea is to memorize for each MBS link the list of examples verifying this link, in the form of a vector V of bits, in which each bit codes an example; we will call this vector V the "truth table" of the link. Thus, if the intersection between two tables is empty, we are able to say that the corresponding links are exclusive ones.

		ex1	*ex2*	*ex3*	
L1 = (X~Y)	=	(*1*	*0*	*1*	)
L2 = (Y~Z)	=	(*0*	*1*	*0*	)
		& ----------	----------	----------	
		(*0*	*0*	*0*	)

=> *The links L1 and L2 are exclusive.*

Furthermore, this method allows us to establish some other relations between the MBS links and therefore, to obtain a more specific generalization. Previously, we have looked at the resolution of the logical equation (Truth table 1 AND Truth table 2 = 0) ; however, it is not the only interesting one. Let two links L1 and L2, and their associated truth tables T1 and T2 :

* If (T1 AND T2 = ø)
  we have the link : (L1 NAND L2)
* If (T1 AND T2 = T2)
  we have the link : (L1 =>L2)
* If (T1 AND T2 = T1)
  we have the link : (L2 => L1)
* If (T1 = T2)
  we have the link : (L1 <=> L2)
* If (T1 XOR T2 = 1)
  we have the link : (L1 XOR L2)

The study of the relations between two MBS links can be generalized to relations between several, in order to bring to the fore some more complex relations such as (L1 & L2 => L3), etc ... This problem has been examined by [Ganascia 87]. Nevertheless, if this kind of information is logically relevant, it often provides some uninteresting information and its computation is time consuming.

* Type dependant links :

The previous links are defined for every kind of variables. However, when the instances of the variables are values, we can add some information to the

generalization formula in order for it to become more specific. The functions which compute this kind of information are type dependant. The usual links can be split in two families : unary links and binary ones. The unary links express a synthesis of the values handled by each variable V. For example, if the instances of V belong to an ordered type, we can add the interval of values to the generalization. The binary links express the observable relations between the values of two variables V1 and V2 belonging to the same type. These links will be created by comparing the definitions of the variables. For instance, in the case of integer values, we can add some links such as "less" or "greater than" between the variables, in the generalization. Here is an example :

E1 = age (a,9) & teeth-nbr (a,28)
E2 = age (b,20) & teeth-nbr (b,32)

G = age (X,A) & teeth-nbr (X,N) &
(A<N) & (A ∈ [9,20]) & (N ∈ [28,32])

3.7 Results of Generalization and Example

In the system KBG, the examples are expressed in the form of a conjunction of instanciated terms. The domain theory is composed of two parts : the first part is the declaration of the data types used, and the second the production rules "IF ... THEN" expressing the relations between the predicates. Moreover, the user can easily define his own types of values by providing to the system the LISP functions expressing the behavior of these items ; he can also put some external procedural calls into the premisses and the conclusions of the rules (for instance, to perform some computation).

In the current implementation, the result of the generalization consists of six different parts :

1 - Generalization : A conjunction of predicates, the arguments of which are variables or constants. These predicates express the common properties and relations found in the example set.

2 - Object matching : The generalization process is based on the matching of the most similar constants (one per example). Thereby, each variable of the generalization's formula corresponds to a list of constants. In this part, we provide all these corresponding substitutions.

3 - Intervals of values : In the case of typed constants, for each corresponding variable we provide in this part the intervals of possible values.

4 - Value links : In the case of typed constants in which the elements are ordered, we can sometimes observe some regularities as "X<Y". This relation means that in all the examples the value of X was always less than the value of Y. In this part, we provide these kind of dependencies.

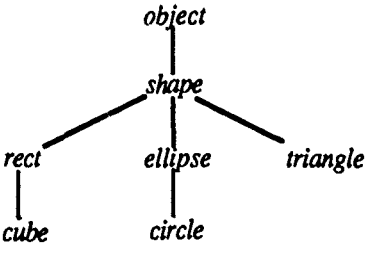
5 - Object links : Implicitly, in order to increase the readability of the result, when two variables always have a different substitution in the examples, their names in the generalization are different. Here we provide the links MBS (May Be the Same) which express that two variables can be either equal or not.

6 - MBS links : Here, we provide the relations between the MBS links (Nand, Xor, ...) described above.

Here is a simple example of a generalization process :

a) Background knowledge :

size : ordered type (micro,small,medium,large,giga);



R1 : IF near (X,Y) THEN near (Y,X) ;
R2 : IF on (X,Y) THEN near (X,Y) ;

b) Examples to generalize :

e1 : rect (a), red (a), size (a, large), circle (b),
on (a, b), size (b, small);
e2 : ellipse (c), red (c), size (c, micro), triangle (d),
on (d, c), size (d, giga);

c) Result of the generalization :

1) Generalization of the examples : (e1 e2)
G = ellipse (C0), shape (C1), on (C1,C0), red (C2)
outsize (C0,TA0), outsize (C1,TA1).
2) Matching : (C0 : b c), (C1 : a d), (C2 : a c),
(TA0 : micros small), (TA1 : large giga).
3) Interval of values : (TA0 ∈ [micro .. small]),
(TA1 ∈ [large .. giga]).
4) Links between values : (TA1 >= TA0).
5) Objects Links (MBS) : (L1 : C2~C0),
(L2 : C2~C1).
6) MBS links : (L1 xor L2).

4 Conclusion

The generalization process can be seen as an operation aiming to extract and to compact the meaningful information held in a example set. The result of this operation can be used by a higher conceptual level mechanism or even by another external system in order to build a knowledge base modeling the studied domain.

However, using generalization as an elementary process requires that it is both accurate and fast. We think that the completion strategy and the algorithms used in our system, allow us to achieve these goals. In the project PERSPICACE, the system KBG was used just as a tool for helping in the construction of the knowledge base. Thus, a large part of the processes were performed manually. At the present time, we are developing two new components, using the generalizer results, in order to increase the automatic parts in the rule construction.

On the one hand, currently when we generalize an example set illustrating one concept, we obtain a large recognition function in which all the terms are not necessarily meaningful ; so, the management of the counter-examples will allow us to perform an "over-generalization" of the learnt knowledge. In this way, we will obtain a more compact and significant information.

On the other hand, the system's capacity to explain its computation is fundamental : indeed, when the expert does not agree, it is often very difficult to find, just by reading the generalization, the modifications to the knowledge representation that are required. We have seen that the generalization process consists of gathering the most similar objects. In our case, the completion of the examples vouches for the fact that the matching will roughly reflect all of the properties expressed in the background knowledge. Thus, the idea is to translate in a readable form the information provided by the similarity computation and to use it to establish a direct dialog between the learning system and the expert. The implementation of explanations will also be the basis on which to elaborate a conceptual clustering of the examples, as a first step towards automatic knowledge base generation.

REFERENCES

[Bollinger 86] BOLLINGER T., "Généralisation en apprentissage à partir d'exemples",Thèse de 3éme cycle, soutenue le 30 janvier 86, Université Paris Sud.

[Cannat 88] CANNAT J.J., VRAIN C., "Machine Learning Applied to Air Trafic Control", Proc. Human-Machine Interaction Artificial Intelligence Aeronautics and Space, pp. 265-274, 28-30 Septembre 88.

[Ganascia 87] GANASCIA J.G., "AGAPE et CHARADE : deux techniques d'apprentissage symbolique appliquées à la construction de bases de connaissances", Thèse d'état 27/4/1987, Université Paris Sud.

[Kodratoff 86] KODRATOFF Y., GANASCIA J.G., "Improving the generalization step in Learning", Machine Learning 2, an Artificial Intelligence Approach, Morgan Kaufmann Publishers, pp. 215-244.

[Kodratoff 88], Y.KODRATOFF, Introduction to Machine Learning, Pitman 1988.

[Kodratoff, Michalski 90] Y.KODRATOFF, R.MICHALSKI, Introduction of Machine Learning 3, an Artificial Intelligence Approach, 1990.

[Michalski 83] MICHALSKI R.S., CARBONNEL J.G., MITCHELL T.M. : "A Theory and Methodology of Inductive Learning",Machine Learning 1, an Artificial Intelligence Approach, Tioga Publishing 83, pp. 83-129.

[Vrain 90] VRAIN C., OGUST : A System Which Learns Using Domain Properties Expressed As Theorems, Machine Learning 3, an Artificial Intelligence Approach, 1990.

Performance Analysis of A Probabilistic Inductive Learning System

Keith C.C. Chan **Andrew K.C. Wong**
PAMI Laboratory
Department of Systems Design Engineering
University of Waterloo
Waterloo, Ontario Canada N2L 3G1

Abstract

This paper presents a new method for acquiring classificatory knowledge by induction. Based on a probabilistic inference technique, this method allows inherent patterns in noisy training instances to be easily detected. A set of classification rules can then be constructed based on these detected patterns. The proposed method has been evaluated by testing it with some real-world data sets and the results show that it out-performs some decision-tree based algorithms both in terms of computational efficiency and classification accuracy.

1. Introduction

Given a collection of objects (events, observations, situations, processes, etc.) that are described in terms of one or more attributes and are preclassified into a number of known classes, the *classification* problem is to find a set of characteristic descriptions for these classes or, equivalently, a procedure for identifying an object as belonging to, or not belonging to a particular one.

Many inductive learning systems have been developed to solve the classification problem and the decision-tree based program ID3 is one of the best known among them [10]. It has been successfully employed for a wide range of applications including several industrial projects [12]. In order to further improve the performance of ID3 when classification have to be made in the presence of uncertainty, a pre-pruning method based on the use of the chi-square test has been employed [12-13]. The problem with the use of this test is that the number of training instances satisfying the path that is being constructed becomes less and less. When the number falls below a certain threshold, the chi-square test cannot be validly used [12]. Therefore, such technique can be employed only when a large training set is available.

The other problem with pre-pruning of decision trees is that pruning decision is made based on local information alone. Due to limited 'lookahead', this process may terminate too early, leaving out important information. To avoid this problem, post-pruning of already-formed decision trees has been proposed. The current post-pruning strategy adopted by ID3 uses a set of production rules equivalent to the tree [12-13]. Based on Fisher's Exact Test, the conditions on the left-hand side of each rule are examined to determine if they are relevant for classification. Conditions that are irrelevant are discarded from a rule. The set of pruned rules are then further simplified by throwing away those whose omission would not lead to more misclassified instances. As observed in [12], this algorithm is rather slow and inefficient when compared to other post-pruning strategies. Furthermore, since hill-climbing strategies are used for pruning the conditions and rules, important information may be lost due to a local optimum [13].

2. An Efficient Classification Method

To efficiently handle uncertainty in classification tasks, we have developed an inductive learning method based on a powerful probabilistic inference technique [1-2]. It can uncover patterns underlying a set of noisy data and is, for this reason, particularly effective in dealing with uncertainty. The method consists of three phases: 1) detection of underlying patterns in training data; 2) construction of classification rules based on the detected patterns; and 3) use of these rules for object classification.

2.1. Discovering Patterns in Noisy Data

Suppose a training set consists of M objects described by n attributes, $Attr_1, \cdots, Attr_n$, so that in an instantiation of object description, $Attr_j$, $1 \leq j \leq n$, takes on $val_j \in domain(Attr_j) = \{v_{jk} | k = 1, \cdots, J\}$. Suppose that the M objects are preclassified into P known classes, c_p, $p = 1, \cdots, P$, the *classification* problem is to find

a set of characteristic descriptions for these classes. To discover patterns underlying this set of noisy training data, attributes important for object class determination need to be identified. To achieve this, many systems use the chi-square test to find attributes that are statistically dependent on the classes [11].

Let o_{ik} be the total number of objects in the training set that belong to c_i, and are characterized by v_{jk} and let e_{ik} be the expected number of such objects under the assumption that the values of $Attr_j$ are randomly distributed in c_i. Then, the chi square statistic can be defined as:

$$X^2 = \sum_{i=1}^{P}\sum_{k=1}^{K}\frac{(o_{ik}-e_{ik})^2}{e_{ik}} = \sum_{i=1}^{P}\sum_{k=1}^{K}\frac{o_{ik}^2}{e_{ik}} - M'. \quad (1)$$

where $M' = \sum_{i,k} o_{ik}$ is less than or equal to M (the total number of objects in the training set) due to the possibility of having missing values in the data. If X^2 is greater than the critical value, $\chi^2_{d,\alpha}$, where $d=(P-1)(J-1)$ is the degree of freedom and α, usually taken to be 0.05 or 0.01, is the significance level (i.e. $(1-\alpha)$% is the confidence level), then one can conclude that $Attr_j$ is statistically dependent on the classes and it is therefore helpful in determining the class membership of an object. Otherwise, there is not enough evidence to support such a conclusion.

However, the chi-square test indicates only that an attribute is important for classification but not which particular value, say v_{jk}, of $Attr_j$ is helpful in determining if an object belongs to a class, say c_i. Such a value can, however, be identified if Pr(object is in $c_i \mid Attr_j = v_{jk}$), is *significantly different* from Pr(object is in c_i) (i.e. o_{ik} should deviate significantly from e_{ik} [1-2]). Since the absolute difference $|o_{ik}-e_{ik}|$ does not provide information on the relative degree of the discrepancy, it needs to be standardized to avoid the influence of the marginal totals. Haberman [7] recommended using the *adjusted residual*:

$$d_{ik} = z_{ik}\Big/\left[(1-\frac{o_{i+}}{M})(1-\frac{o_{+k}}{M})\right]^{1/2} \quad (2)$$

o_{i+} being the total number of objects in the training set that are in c_i, and o_{+k}, the total number of training objects having the characteristic v_{jk} [7].

For $p = 1, 2, \cdots, P$, and $k = 1, 2, \cdots, J$, if the absolute value of an adjusted residual, say d_{ik}, is greater than 1.96, the 95 percentiles of the normal distribution (or 99 percentiles or higher for a greater confidence level), we can conclude that the discrepancy between o_{ik} and e_{ik} (i.e. between Pr(object is in $c_i \mid Attr_j = v_{jk}$) and Pr(object is in c_i)) is significantly different and therefore v_{jk} is important for the classification of the objects it characterizes. The sign of d_{ik} is also important. A $d_{ik} \geq +1.96$ indicates that the presence of v_{jk} is a relevant value of c_i whereas a $d_{ik} \leq -1.96$ indicates it is more unlikely for an object characterized by v_{jk} to be a member of c_i than other classes.

Values of $Attr_j$ showing no correlation with any class yield no classificatory information. These values are considered as *irrelevant* for learning. Their inclusion may cause overfitting and the generation of misleading classification rules. Hence, they are discarded from further analysis.

2.2. Construction of Classification Rules

To use the detected relevant values for each object class, they are explicitly represented in the classification rules. Suppose that v_{jk} is a relevant value of c_i, the following rule is constructed.

> If $Attr_j$ of an object is v_{jk} then that this object belongs to c_i is with weight of evidence
> $W(Class=c_i/Class\neq c_i \mid Attr_j=v_{jk})$,

where $W(Class=c_i/Class\neq c_i \mid Attr_j=v_{jk})$ measures the amount of positive or negative evidence provided by v_{jk} supporting or refuting an object that it characterizes to be classified into c_i.

The derivation of $W(Class=c_i\ /Class\neq c_i \mid Attr_j=v_{jk})$ is based on an information theoretic measure known as the mutual information. The mutual information between c_i and the relevant feature, v_{jk}, is defined as [8, 16]:

$$I(Class=c_i; Attr_j=v_{jk}) = \log\frac{Pr(Class=c_i \mid Attr_j=v_{jk})}{Pr(Class=c_i)}. \quad (3)$$

so that $I(Class=c_i : Attr_j=v_{jk})$ is positive if and only if $Pr(Class=c_i \mid Attr_j=v_{jk}) > Pr(Class=c_i)$ otherwise it is either negative or has a value 0. Based on this measure, the weight of evidence provided by v_{jk} in favor of an object being a member of c_i as opposed to its not being a member of the class, can be defined as follows [8]:

$W(Class=c_i/Class\neq c_i \mid Attr_j=v_{jk})$

$$= I(Class=c_l:Attr_j=v_{jk}) - I(Class\neq c_l:Attr_j=v_{jk})$$

$$= \log \frac{Pr(Attr_j=v_{jk}|Class=c_l)}{Pr(Attr_j=v_{jk}|Class\neq c_l)}. \quad (4)$$

In other words, W may be interpreted as the difference in information about an object being in c_l compared to other classes given that it is characterized by v_{jk} [8]. W is therefore, in its technical sense, the log of the likelihood ratio. It should be noted that the likelihood ratio is well known among Bayesian statisticians. It is, perhaps for this reason, that it is also quite popular among AI researchers as a scheme for uncertainty representation. For example, Duda et. al found its use quite convenient for plausible reasoning in Prospector [5]. Schlimmer et. al also adopted it in the STAGGER system [14].

The class description of a class c_l can be considered as the subset of classification rules whose conclusion parts predict c_l. These rules describe each class of training objects probabilistically. It should be noted that, like the decision-tree based inductive learning systems which determine the *information gain* [10] of each attribute independent of the others at a node, the proposed method also evaluates the weight of evidence provided by an attribute value for classification independently. Hence, each classification rule represents the amount of evidence provided by a single attribute value for or against an object's being assigned to a certain class. Conjunctive rules can, however, be formed by combining the *first-order* rules. For example, consider the following rules:

1. If $Attr_i=v_i$ then $Class=c$ with $W=w_i$.
2. If $Attr_j=v_j$ then $Class=c$ with $W=w_j$.

These rules can be combined to form Rule No. 3:

3. If $Attr_i=v_i$ and $Attr_j=v_j$ then $Class=c$ with $W=w_i+w_j$.

In other words, high-order conjunctive rules can be combined to form low-order ones if the conclusion parts of the low-order rules are the same. If the condition parts of two or more rules involve the same attribute, they can be combined to form disjunctive rules. For example, consider the following rules:

4. If $Attr_i=v_{i_1}$ then $Class=c$ with $W=w_{i_1}$.
5. If $Attr_i=v_{i_2}$ then $Class=c$ with $W=w_{i_2}$.

These rules can be combined to form Rule No. 6:

6. If $Attr_i=v_{i_1}$ or $Attr_i=v_{i_2}$ then $Class=c$ with $W=w_{i_1}$ or $W=w_{i_2}$.

Though not explicitly represented as high-order rules, the combination of the low-order ones are necessary in order to determine the class membership of an object (Section 2.3). As will be described later in Section 3 and 4, based on this rule combination technique, the proposed method can be shown to perform better, both in terms of accuracy and learning efficiency, than common classification techniques.

Other than combining low-order rules in the above manner, high-order rules can also be formed by considering attribute values together, instead of separately, in determining their relevancy for classification (Section 2.1). The details of how this can be done will be discussed in a separate paper.

2.3. Classification of Objects

Suppose that an object *obj* described by n characteristics $val_1, \cdots, val_j, \cdots, val_n$ is given. By matching each attribute value of *obj* against each classification rule in turn, the classes it may be assigned to can be determined. However, since the description of *obj* may match partially with that of more than one class of objects, it may be classified into different classes based on its different characteristics. As is discussed in the last section, since each of the attribute values of *obj* that matches the classification rules can be considered as providing some evidence for or against the assignment of *obj* to those classes predicted by the rules, its classification can be made based on a measure that combines these pieces of evidence together. Its value should increase with the strength and the number of pieces of positive evidence supporting a specific class assignment for *obj* and decreases vice versa. One of such measure that possess such property is proposed here. It estimates quantitatively the various pieces of evidence, provided by $val_1, \cdots, val_n$ in favor of *obj* being classified into c_{obj} as opposed to being classified into other classes. It is defined as:

$$W(C_{obj}=c_{obj}/C_{obj}\neq c_{obj}|val_1, \cdots, val_n)$$

$$= \log \frac{Pr(C_{obj}=c_{obj}|val_1, \cdots, val_n)}{Pr(C_{obj}=c_{obj})}$$

$$- \log \frac{Pr(C_{obj}\neq c_{obj}|val_1, \cdots, val_n)}{Pr(C_{obj}\neq c_j)},$$

$$= \log \frac{Pr(val_1, \cdots, val_n|C_{obj}=c_{obj})}{Pr(val_1, \cdots, val_n|C_{obj}\neq c_{obj})} \quad (5)$$

Suppose that, of the n attributes that describe *obj*, only m $(m\leq n)$ of them, $val_{[1]}, \cdots, val_{[j]}, \cdots, val_{[m]}$, $val_{[j]} \in \{val_j|j = 1, \cdots, n\}$, are found to match one or more classification rules,

then the weight of evidence can be simplified into:

$$W(C_{obj}=c_{obj}/C_{obj}\neq c_{obj}|val_1, \cdots, val_n) = W(C_{obj}=c_{obj}/C_{obj}\neq c_{obj}|val_{[1]}, \cdots, val_{[m]}) \quad (6)$$

If there is no a priori knowledge concerning the interrelation of the attributes in the problem domain, then this weight provided by all the attribute values of *obj* in favor of it being assigned to c_{obj} as opposed to being assigned to other classes is equal to the sum of the weights provided by each individual attribute value of *obj* that is relevant for classifying it to c_{obj} [1]:

$$W(C_{obj}=c_{obj}/C_{obj}\neq c_{obj}|val_{[1]}, \cdots, val_{[m]}) = \sum_{j=1}^{m} W(C_{obj}=c_{obj}/C_{obj}\neq c_{obj}|val_{[j]}). \quad (7)$$

In brief, the classification strategy can be summarized as follows. Given an object *obj*, the set of classification rules is searched to determine which class descriptions are partially matched by that of *obj*. If a value satisfies the condition part of a rule, the positive and negative evidence provided by the attribute values of *obj* are quantitatively measured and combined for comparison. *obj* is assigned to c_p when it has the largest partial match, that is:

$$W(C_{obj}=c_p/C_{obj}\neq c_p|val_{[1]} \cdots, val_{[m]}) > W(C_{obj}=c_h/C_{obj}\neq c_h|val_{[1]}, \cdots, val_{[m]}), \quad h=1, 2, \cdots, P' \text{ and } h\neq p \quad (8)$$

where P' $(\leq P)$ denotes the number of classes that are partially matched by the attribute values of *obj*.

It should be noted that if two different plausible values have the same greatest weight of evidence, there may be more than one plausible class assignment for *obj*. Furthermore, if there is no evidence for or against any specific class assignment, classification may either be refrained to avoid furnishing of an inaccurate assignment or that *obj* can be assigned to the class to which the majority of training objects belong. If it happens that there is no relevant value for determining the class membership of *obj*, then either the training data is completely non-deterministic or there is insufficient training instances for the learning process.

3. Experimental Results

To evaluate the performance of the proposed probabilistic inductive learning method and compare it with other classification methods, the following data sets are used:

1. Dermatoglyphic Data – Over decades, numerous articles have been published reporting clinical significance of finger prints and palm patterns in cogenital disease identification [4, 15, 17]. For an experiment with the dermatoglyphic data, the finger-print patterns, the *atd angle* and the *ab ridge-count* of both the right and left hands of 126 subjects were obtained [17] (Fig. 1). These 126 subjects had been divided into three groups. The first of these consisted of 51 myelomeningocele patients drawn from a spinal dysfunction clinic. The second and third groups consisted of 40 and 35 normal subjects, respectively, that show different dermatoglyphic patterns [17].
2. Medical Data I – For further performance evaluation, we used two sets of medical data obtained from an Intensive Care Unit (ICU) [9]. The first set consisted of 120 patient records each of which was characterized by 12 attributes representing different symptoms derived from the monitoring of the Central Nervous System, the Respiratory System, the Cardiovascular System, the Skin Signs and the Renal System of the patient [9]. Based on the attributes, the patients were classified into two groups by the physicians – those that had to be placed under intensive care and those that could be discharged to the main floor. The records of 66 patients were available for the first group and 54 for the second.
3. Medical Data II - The second set of medical data obtained from the same source consist of 99 records characterized from the same 12 attributes [9]. Based on these attributes, each patient was diagnosed, by a group of physicians, into one of four disease types [9]: (i) chest disease; (ii) abdominal disease; (iii) cardiac disease; and (iv) neurological disease. In this experiment, the records of 15 patients were available for Group 1, 33 for Group 2, 25 for Group 3 and 26 for Group 4.

For performance evaluation, the classification accuracy of the following systems for the above data sets were determined: (i) the ID3 method which builds a decision tree for classification based on the TDIDT (Top-Down Induction of Decision Tree) algorithm described in [10]; (ii) the ID3 method with pre-pruning which terminates the branching process at a node when the attribute associated with the node is tested (by the chi-square test) to be statistically independent of the classes [11]; (iii) the ID3 method with post-pruning which constructs a set of rules equivalent to a decision tree and then simplifies them by testing each term of each rule against the class, predicted by the conclusion of the rule, for statistical dependence (using Fisher's

Exact Test) [12-13]; (iv) the Bayesian Classifier which computes the class conditional probability for every class by assuming, due to small sample size for training, that all the attributes are independent and then selects the class with the highest probability; (v) the Default or Simple Majority Classifier which simply assigns the most commonly occurring class in the training set to all objects that are to be classified, with no reference to their attributes; and (vi) the proposed method without rejection (i.e, an object is assigned to the most commonly occurring class when there is not enough evidence supporting it to be assigned to any specific one). In the evaluation, ten different randomly selected training (70% of total available data) and testing samples were generated for each of the three sets of data. The results, averaged over the ten randomly chosen training and testing samples, are given in Table 1.

As is expected, these results indicate that the classification accuracy of the Simple Majority method, which served as a reference point for evaluating the various methods, was the poorest. The Bayesian Classifier performed much better than the Simple Majority method. However, it was, in general, not as good as the decision-tree-based algorithms except in the experiment with the second set of medical data.

Of the three ID3- or decision-tree-based methods, the use of a pre-pruning strategy during the construction of decision trees did not seem to improve the classification accuracy. In fact, it consistently made more errors than the simple ID3-algorithm in which no pre-pruning was performed. This was due to the use of the chi-square test in deciding whether pruning at a node should be terminated. In order for the assumptions behind the chi-square test to be valid, a very large training set for each class of objects have to be available. This is because, as a decision tree is being built, the set of training samples is always being subdivided in the hope that no single exception remains. Therefore, the further down the tree, the smaller is the resulting subset of training sample. Since, according to [11], the expected frequencies of each cell in the contingency table constructed for a chi-square test should be greater than four, unavailability of a large training set for each class often result in the branching process terminating too early.

Recently, in the statistics community, it is discovered that the restriction for the expected frequency in each cell to be at least four is too conservative. It is suggested that, at least for test conducted at a confidence level of 95%, the minimum expected cell frequencies can be approximately 1.0 [6]. In fact, the test for the ID3-algorithm with pre-pruning was performed under such an assumption. Unfortunately, it did not improve the classification accuracy of the algorithm by much.

As for the ID3 method without pruning and that with post-pruning, they are comparable in performance in the sense that the former is more accurate in experiments with the first set of medical data but not as accurate as in the other experiments.

Of all the six methods, the percentage of correct classification of the proposed method was the highest in all experiments. Direct comparison between proposed method and the other decision-tree pruning methods are yet unavailable. However, while ID3 with post-pruning have been shown to have superior performance when compared to the others [12], we only compared the proposed method with ID3.

4. Discussions and Summary

In evaluating the performance of a learning method, it is often necessary to consider its decision accuracy, the amount of training required to achieve a specific level of performance, as well as how costly that training is. One measure of training effort is the complexity of the method [3].

Let M denote the size of the training set, n the total number of attributes that describe each of the training instances. The critical component of the ID3 algorithm is the process of selecting a test attribute on which to branch. Each such choice involves the following operations [3]:

1. For each attribute, example counts are put in an array, indexed by class and attribute. This takes time $O(n \bullet M)$;
2. The entropy function is calculated for each attribute, taking time $O(n)$;
3. Once the best attribute is found, the examples are divided into J different sets (where J is the total number of values that the best attribute takes on). This takes time $O(M)$.

Therefore, the overall time for a single attribute choice at each node in a decision tree is $O(n \bullet M)$. The time taken to construct the complete tree depends very much on the structure of the tree. In general, it is $O(n \bullet M \bullet$ (Total no. of nodes in the tree)).

As for the ID3 algorithm with pre-pruning, the need to test for statistical dependence between each attribute and the classes at each node does not affect the complexity for the critical component of the ID3 algorithm. In fact, since a smaller subtree with fewer number of nodes is usually obtained as a result of tree-pruning, the overall rate of learning may even be improved.

The ID3 algorithm with post-pruning requires a complete decision tree to be built first. The complexity for this process is $O(n \bullet M \bullet$ (Total no. of nodes in the tree)). The second part of the post-pruning algorithm involves constructing a set of rules equivalent to the decision tree. Each path in the tree is turned into a rule with the number of conditions on the left-hand side of the rule equals to the path length. The conditions of each rule are then examined in turn to determine, by Fisher's Exact Test for statistical independence in two-by-two contingency tables, the ones that are relevant for the classification process. Conditions that are irrelevant are discarded from a rule. The time complexity of this process is, once again, very much dependent on the structure of the tree and, therefore, the rules constructed from it. In particular, it is dependent on the number of rules and the number of conditions on their left-hand sides. The process takes time $O(M \bullet$ (Total no. of rules)$\bullet$ (Average no. of conditions per rule)).

To further simplify the set of pruned rules, each of them is omitted in turn so as to determine how well the rest of the rules perform on the training instances. If there are rules whose omission would not lead to an increase in the number of misclassified instances or would even reduce it, then the least useful one is discarded and the process is repeated until no such rule can be found. The time complexity of this process is dependent on the number of rules that have to be deleted from the resulting rule set after the rule simplification process. This process takes time $O(M \bullet$ (Total no. of rules)).

The time complexity of the proposed classification method to generate the contingency tables to search for significant adjusted residuals is $O(n \bullet M)$. Being independent of the run time of the classification rule generated, this learning method is substantially faster than the decision-tree-based algorithms whose efficiency depends very much on the structure of the decision trees. In addition, unlike the decision-tree-based algorithms in which the basic operation is repeatedly applied, the basic operation of the proposed method, i.e., the analysis of residuals, is performed only once. Since the residuals can be examined independently of each other, the proposed learning method has been employed in the implementation of an artificial Neural network which has the advantage of being very fast in its training process as well as being able to provide a facility for direct analysis of internal associations. This network has been tested on simulated and real-world data with performance superior to a Back Propagation Network [18].

In summary, we have presented an efficient probabilistic inductive learning method for solving classification problem even when (i) the data contains inaccurate, incomplete and inconsistent values; (ii) the assumption concerning any specific mathematical model for the data cannot be made; (iii) the data is of high dimensionality and (iv) the training sample size is relatively small. The proposed method has been implemented and tested using real-life data sets. The test results showed that the proposed method consistently outperformed, in terms of accuracy and computational efficiency, many commonly used methods that were developed to handle uncertainty in classification tasks.

5. References

[1] Chan, K.C.C. and Wong, A.K.C., 'PIS: A Probabilistic Inference System', to appear in the *Proceedings of the 9th International Conference on Pattern Recognition*, Rome, Italy, November, 1988.

[2] Chan, K.C.C., and Wong, A.K.C., 'Automatic Construction of Expert Systems from Data: A Statistical Approach', *Proceedings of the IJCAI'89 Workshop on Knowledge Discovery in Databases*, Detroit, Michigan, Aug., 1989.

[3] Clark, P., and Niblett, T., 'The CN2 Induction Algorithm', *Machine Learning*, **3**, 1989, pp.261-283.

[4] Cummins, H., 'Dermatoglyphic Stigmata in Mongolian Idiocy', *Anatomical Record*, **64** (Suppl. 2), 11, 1936.

[5] Duda, R., Gaschnig, J., and Hart, P., 'Model Design in The Prospector Consultant System for Mineral Exploration', in Michie, D., (ed.), *Expert Systems in the Micro-Electronic Age*, Edinburgh University Press, Edinburgh, 1979.

[6] Fienberg, S.E., *The Analysis of Cross-Classified Categorical Data*, 2nd Edition, MIT Press, Cambridge, MA 1980.

[7] Haberman, S.J., 'The Analysis of Residuals in Cross-Classified Tables', *Biometrics*, **29**, 1973, pp. 205-220.

[8] Osteyee, D.B., and Good, I.J., *Information, Weight of Evidence, the Singularity between Probability Measures and Signal Detection*, Springer-Verlag, Berlin, 1974.

[9] Pao, Y.H., and Hu, C.H., 1984, 'Processing of Pattern-based Information: Part I: Inductive Inference Methods Suitable for Use in Pattern Recognition and Artificial Intelligence', in Tou, J.T., *Advances in Information Systems Sciences*, Vol. 9, Plenum Press.

[10] Quinlan, J.R., 'Learning Efficient Classification Procedures and Their Application to

Chess End-Games', in R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), *Machine Learning: An Artificial Intelligence Approach*, Tioga, Palo Alto, CA, 1983.

[11]
Quinlan, J.R., 'The Effect of Noise on Concept Learning', in R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), *Machine Learning: An Artificial Intelligence Approach*, Vol.2, Tioga, Palo Alto, CA, 1986.

[12]
Quinlan, J.R., 'Simplifying Decision Trees', *International Journal Man-Machine Studies*, **27**, 1987, pp.221-234.

[13]
Quinlan, J.R., 'Generating Production Rules From Decision Trees', *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, 1987, pp.304-307.

[14]
Schlimmer, J.C., and Granger, R.H., 'Incremental Learning from Noisy Data', *Machine Learning*, **1**, 1986, pp.317-354.

[15]
Steg, N.L., Wong, A.K.C., Vogel, M.A., Casey, P.A., Constance, E.C., and Wang, D.C.C., 'Dermatoglyphic Analysis by PAC Discriminant Tree Classification', *Birth Defects: Original Article Series*, **15**, 6, 1979, pp.149-162.

[16]
Wang, D.C.C., and Wong, A.K.C., 'Classification of Discrete Data with Feature Space Transformation', *IEEE Transactions on Automatic Control*, **AC-24**, 3, 1979.

[17]
Wong, A.K.C., Vogel, M.A., and Wang, D.C.C., 'Computer Augmented Methodologies for Classifying Clinical Data', Internal Report, Biomedical Information Processing Program, Carnegie-Mellon University, Pittsburgh, 1975.

[18]
Wong, A.K.C., Veith, J.O., and Chan, K.C.C., 'A Neural Network Implementation of A Statistical Classifier', submitted for publication, 1990.

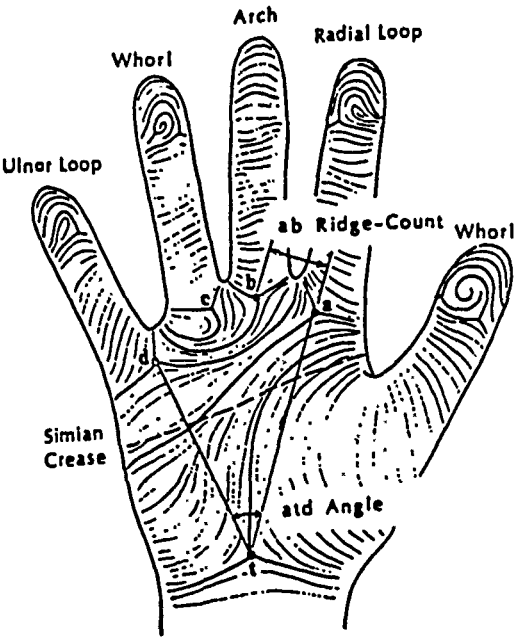


Fig. 1. Dermatoglyphic Patterns [15]

Table 1. Performance (% Correct) Comparison of Different Classification Methods

Data	Classification Methods					
	ID3 without pruning	ID3 with pre-pruning	ID3 with post-pruning	Bayesian Classifier	Simple Majority	PIT-Based APACS
Dermatoglyphic Data	51.0%	48.2%	55.0%	44.7%	37.5%	68.2%
Medical data I	89.2%	82.0%	85.0%	79.2%	51.1%	91.1%
Medical data II	79.5%	59.5%	87.0%	81.0%	35.0%	94.0%

A Comparative Study of ID3 and Backpropagation for English Text-to-Speech Mapping

Thomas G. Dietterich
tgd@cs.orst.edu

Hermann Hild
s_hild@iravi.ira.uka.de

Ghulum Bakiri
haya@cs.orst.edu

Department of Computer Science
Oregon State University
Corvallis, OR 97331-3902

Abstract

The performance of the error backpropagation (BP) and ID3 learning algorithms was compared on the task of mapping English text to phonemes and stresses. Under the distributed output code developed by Sejnowski and Rosenberg, it is shown that BP consistently out-performs ID3 on this task by several percentage points. Three hypotheses explaining this difference were explored: (a) ID3 is overfitting the training data, (b) BP is able to share hidden units across several output units and hence can learn the output units better, and (c) BP captures statistical information that ID3 does not. We conclude that only hypothesis (c) is correct. By augmenting ID3 with a simple statistical learning procedure, the performance of BP can be approached but not matched. More complex statistical procedures can improve the performance of both BP and ID3 substantially. A study of the residual errors suggests that there is still substantial room for improvement in learning methods for text-to-speech mapping.

1 Introduction

The task of mapping English text into speech is quite difficult (see Klatt, 1987). One particularly difficult step involves mapping words (i.e., strings of letters) into strings of phonemes and stresses. In this paper, we compare two machine learning algorithms applied to the task of learning this text-to-speech mapping. We employ the formulation developed by Sejnowski and Rosenberg (1987) in their widely known work on NETTALK.

Let L be the set of 29 symbols comprising the letters **a**–**z**, and the comma, space, and period (in our data sets, comma and period do not appear). Let P be the set of 54 English phonemes and S be the set of 6 stresses employed by Sejnowki and Rosenberg. The task is to learn the mapping $f : L^* \longrightarrow P^* \times S^*$. Specifically, f maps from a word of length k to a string of phonemes of length k and a string of stresses of length k. For example,

```
f("lollypop") = ("lal-ipap", ">1<>0>2<").
```

Notice that letters, phonemes, and stresses have all been aligned so that silent letters are mapped to the silent phoneme /-/.

As defined, f is a very complex discrete mapping with a very large range. If we assume no word contains more than 28 letters, this range would contain more than 10^{70} elements. Existing learning algorithms focus primarily on learning Boolean concepts—that is, functions whose range is the set $\{0,1\}$. Such algorithms cannot be applied directly to learn f.

Fortunately, Sejnowski and Rosenberg developed a technique for converting this complex learning problem into the task of learning a collection of Boolean concepts. They begin by reformulating f to be a mapping g from a seven-letter window to a single phoneme and a single stress. For example, the word "lollypop" would be converted into 8 separate 7-letter windows:

```
g("___loll") = ("l", ">")
g("__lolly") = ("a", "1")
g("_lollyp") = ("l", "<")
g("lollypo") = ("-", ">")
g("ollypop") = ("i", "0")
g("llypop_") = ("p", ">")
g("lypop__") = ("a", "2")
g("ypop___") = ("p", "<")
```

The function g is applied to each of these 8 windows, and then the results are concatenated to obtain the phoneme and stress strings. This mapping function g now has a range of 324 possible phoneme/stress pairs, which is a substantial improvement.

Finally, Sejnowski and Rosenberg code each possible phoneme/stress pair as a 26-bit string, 21 bits for the phoneme and 5 bits for the stress. Each bit in the code corresponds to some property of the phoneme or stress. This converts g into 26 separate Boolean functions, $h_1, \ldots, h_{26}$. Each function h_i maps from a seven-letter window to the set $\{0,1\}$. To assign a phoneme and stress to a window, all 26 functions are evaluated to produce a 26-bit string. This string is then mapped to the nearest of the 324 bit strings representing legal phoneme/stress pairs. We used the Hamming distance

between two strings to measure distance. (Sejnowski and Rosenberg used the angle between two strings to measure distance, but they report that the Euclidean distance metric gave similar results. In tests with the Euclidean metric, we have obtained results identical to those reported in this paper.)

With this reformulation, it is now possible to apply Boolean concept learning methods to learn the h_i. However, the individual h_i must be learned extremely well in order to obtain good performance at the level of entire words. This is because errors aggregate. For example, if each h_i is learned so well that it is 99% correct and if the errors among the h_i are independent, then the 26-bit string will be correct only 77% of the time. Because the average word has about 7 letters, whole words will be correct only 16% of the time.

In the remainder of this paper, we describe a series of experiments comparing the performance of the error backpropagation algorithm (BP) to the decision-tree learning algorithm ID3. We begin by comparing BP and ID3 on the task described above. Having established that BP significantly outperforms ID3 on this task, we formulate three hypotheses to explain this difference. We test these hypotheses by performing additional experiments. These experiments demonstrate that ID3, combined with some simple statistical learning procedures, can nearly match the performance of BP. Finally, we present data suggesting that there is still substantial room for improvement of learning algorithms for text-to-speech mapping.

2 A Simple Comparative Study

In this study, ID3 and BP were both applied to the learning task described above. We begin by briefly reviewing these two learning algorithms and the data set.

2.1 The Algorithms

ID3 is a simple decision-tree learning algorithm developed by Ross Quinlan (1983; 1986b). The version we employed used the information gain criterion to choose which feature to place at the root of each decision tree (and subtree). We did not employ windowing (Quinlan, 1983), CHI-square forward pruning (Quinlan, 1986a), or any kind of reverse pruning (Quinlan, 1987). We did apply one simple kind of forward pruning to handle inconsistencies in the training data: If all training examples agreed on the value of the chosen feature, then growth of the tree was terminated in a leaf and the class having more training examples was chosen as the label for that leaf (in case of a tie, the leaf is assigned to class 0).

To apply ID3 to this task, the algorithm must be executed 26 times—once for each mapping h_i. Each of these executions produces a separate decision tree. The seven-letter window was represented as the concatenation of seven 29-bit strings. Each 29-bit string represents a letter (one bit for each letter, period, comma, and blank), and hence, only one bit is set to 1 in each 29-bit string. This produces a string of 203 bits for each window.

The error backpropagation algorithm (Rumelhart, Hinton & Williams, 1986) is widely applied to train artificial neural networks. We replicated the network architecture and training procedure employed by Sejnowski and Rosenberg (1987). This network is a fully-connected feed-forward network containing 203 input units, 120 hidden units, and 26 output units (one for each mapping h_i). We employed the same input and output encodings described above.

Unlike ID3, it is only necessary to apply BP once, because all 26 output bits can be learned simultaneously. Indeed, the 26 outputs all share the collection of 120 hidden units, which may allow them to be learned more accurately. However, while ID3 is a batch algorithm that processes the entire training set at once, BP is an incremental algorithm that makes repeated passes over the data. Each complete pass is called an "epoch." During an epoch, the training examples are inspected one-at-a-time, and the weights of the network are adjusted to reduce the squared error of the outputs. We used a learning rate of .25 and a momentum term of .9. The weights of the network were initialized to random values between $-.3$ and $+.3$. In all cases, we trained for 30 epochs, since this was the training regime followed by Sejnowski and Rosenberg. We used the implementation provided with (McClelland and Rumelhart, 1988).

Because the outputs from BP are floating point numbers between 0 and 1, we had to adapt the Hamming distance measure when mapping to the nearest legal phoneme/stress pair. We used the following distance measure: $d(\mathbf{x},\mathbf{y}) = \sum_i |x_i - y_i|$. This reduces to the Hamming distance when $\mathbf{x}$ and $\mathbf{y}$ are Boolean vectors.

2.2 The Data Set

Sejnowski and Rosenberg provided us with a dictionary of 20,003 words and their corresponding phoneme and stress strings. This dictionary was randomly partitioned into a testing set of 1000 words, and a training set of 19,003 words. This training set was further subdivided to extract smaller training sets of 1000, 800, 400, 200, 100, and 50 words. Each smaller training set was extracted by randomly sampling from the next larger set.

2.3 Results

Table 1 shows percent correct (over the 1000-word test set) as a function of the size of the training set for words, letters, phonemes, and stresses. Virtually every difference in the table at the word, letter, phoneme, and stress levels is statistically significant (using the test for the difference of two proportions). Hence, we conclude that there is a substantial difference in performance between ID3 and BP on this task.

To take a closer look at the performance difference, we can study exactly how each of the 7,242 windows in the test set are handled by each of the algorithms. Table 2 categorizes each of these windows according to whether it was correctly classified by both algorithms, by only one of the algorithms, or by neither one.

Table 1: Percent correct over 1000-word test set

Sample Size	Method	Level of Aggregation (% correct)				
		Word	Letter	Phoneme	Stress	Bit (mean)
50	ID3	0.8	41.5	60.5	60.1	93.1
	BP	1.8*	48.4***	59.4	72.9***	93.5
100	ID3	2.0	47.3	64.1	65.8	94.0
	BP	3.7*	55.2**	66.1**	75.5**	94.4
200	ID3	4.4	56.6	70.5	72.2	95.1
	BP	6.0	61.4***	71.9*	78.6***	95.3
400	ID3	6.2	58.7	73.7	72.1	95.5
	BP	10.5***	65.7***	76.0***	79.9***	95.9
800	ID3	9.6	63.8	77.8	75.6	96.2
	BP	12.2*	68.7***	78.9	80.7***	96.3
1000	ID3	9.6	65.6	78.7	77.2	96.4
	BP	14.7***	70.9***	81.1***	81.4***	96.6

Difference in the cell significant at $p < .05^{*}$, $.01^{**}$, $.001^{***}$

Table 2: Classification of test set windows by ID3 and Backpropagation, decoding to nearest legal phoneme and stress.

		Back Propagation		
		Correct	Incorrect	
ID3	Correct	4231	520	4751
	Incorrect	907	1584	2491
		5138	2104	

The table shows that the windows correctly learned by BP do not form a superset of those learned by ID3. Instead, the two algorithms share 4,231 correct windows, and then each algorithm correctly classifies several windows that the other algorithm gets wrong. The net result is that BP classifies 387 more windows correctly than does ID3. This shows that the two algorithms, while they share substantial overlap, have learned substantially different text-to-speech mappings.

The information in this table can be summarized as a correlation coefficient. Specifically, let X_{ID3} (X_{BP}) be a random variable that is 1 iff ID3 (BP, respectively) makes a correct prediction at the letter level. In this case, the correlation between X_{ID3} and X_{BP} is .5508. If all four cells of Table 2 were equal, the correlation coefficient would be zero.

A weakness of Table 1 is that it shows performance values for one particular choice of training and test sets. We have replicated this study four times (for a total of 5 independent trials). Table 3 shows the average performance of these 5 runs (each, of course, on a different randomly-drawn 1000-word test set). All differences are significant below the .0001 level using a t-test for paired differences.

In the remainder of this paper, we will attempt to understand the nature of the differences between BP and ID3. Our main approach will be to experiment with modifications to the two algorithms that enhance or eliminate the differences between them. All of these experiments are performed using only the training set and test set from Table 1.

3 Three Hypotheses

What causes the differences between ID3 and BP? We have three hypotheses:

Hypothesis 1: Overfitting. ID3 has overfit the training data, because it seeks complete consistency. This causes it to make more errors on the test set.

Hypothesis 2: Sharing. The ability of BP to share hidden units among all of the h_i may allow it to reduce the aggregation problem at the bit level.

Hypothesis 3: Statistics. The numerical parameters in the BP network allow it to capture statistical information that is not captured by ID3.

These hypotheses are neither exclusive nor exhaustive.

The following two subsections present the experiments that we performed to test these hypotheses.

3.1 Tests of Hypothesis 1 (Overfitting)

The tendency of ID3 to overfit the training data is well established in cases where the data contain noise. Three basic strategies have been developed for addressing this problem: (a) criteria for early termination of the tree-growing process, (b) techniques for pruning trees to remove overfitting branches, and (c) techniques for converting the decision tree to a collection of rules. We implemented and tested one method for each of these strategies. Table 4 summarizes the results.

The first row repeats the basic ID3 results given above, for comparison purposes. The second row shows the effect of applying a χ^2 test (at the .90 confidence level) to decide whether further growth of the decision tree is statistically justified (Quinlan, 1986a). As other authors have reported (Mooney et

Table 3: Average percent correct (1000-word test set) over five trials.

Sample Size	Method	Level of Aggregation (% correct)				
		Word	Letter	Phoneme	Stress	Bit (mean)
1000	ID3	10.2	65.2	79.1	76.5	96.4
	BP	14.3*	70.7*	81.2*	81.0*	96.6*

Difference in the cell significant at $p < .0001$*

Table 4: Results of applying three overfitting-prevention techniques.

Method	Data set	Level of Aggregation (% correct)				
		Word	Letter	Phoneme	Stress	Bit (mean)
(a) ID3 (as above)	TEST:	9.6	65.6	78.7	77.2	96.4
(b) ID3 (χ^2 cutoff)	TEST:	9.1	64.8	78.4	77.1	96.4
(c) ID3 (pruning)	TEST:	9.3	62.4	76.9	75.1	96.1
(d) ID3 (rules)	TEST:	8.2	65.1	78.5	77.2	96.4

al., 1989), this hurts performance in the Nettalk domain. The third row shows the effect of applying Quinlan's technique of reduce-error pruning (Quinlan, 1987). Mingers (1989) provides evidence that this is one of the best pruning techniques. For this row, a decision tree was built using the 800-word training set, and then pruned using the additional words from the 1000-word training set that do not appear in the 800-word training set. Other trials using words from a 1600-word training set produced similar results. Finally, the fourth row shows the effect of applying Quinlan's method for converting a decision tree to a collection of rules. Quinlan's method has three steps, of which we performed only the first two. First, each path from the root to a leaf is converted into a conjunctive rule. Second, each rule is evaluated to remove unnecessary conditions. Third, the rules are combined, and unnecessary rules are eliminated. The third step was too expensive to perform on this rule set, which contains 6,988 rules.

None of these techniques improved the performance of ID3 on this task. This suggests that Hypothesis 1 is incorrect: ID3 is not overfitting the data in this domain. This makes sense, since the only source of "noise" in this domain is the limited size of the 7-letter window and the existence of a small number of words like "read" that have more than one correct pronunciation. Seven-letter windows are sufficient to correctly classify 98.5% of the words in the 20,003-word dictionary.

3.2 A Test of Hypothesis 2 (Sharing)

To test the sharing hypothesis, we attempted to train 26 independent networks, each having only one output unit, to learn the h_t mappings. If Hypothesis 2 is correct, then, because there is no sharing among these separate networks, we should see a drop in performance compared to the single network with shared hidden units. Furthermore, the decrease in performance should decrease the differences between BP and ID3.

Surprisingly, we were unable to train successfully the separate networks to the target error level on any training set other than the 50-word set. For the 100-word training set, for example, the individual networks often converged to local minima (even though the 120-hidden-unit network had avoided these minima). This shows that even if shared hidden units do not aid classification performance, they certainly aid the learning process!

As a consequence of this training problem, we are able to report results for only the 50-word training set. Table 5 shows the performance of these 26 networks on the training and test sets. Performance on the training set is virtually identical to the 120-hidden-unit network, which shows that our training regime was successful. Performance on the test set, however, shows a loss of performance when there is no sharing of the hidden units among the output units. Hence, it suggests that Hypothesis 2 is at least partially correct. However, examination of the correlation between ID3 and BP indicates that this is wrong. The correlation between X_{ID3} and X_{BP1} (i.e., BP on the single network) is .5167, whereas the correlation between X_{ID3} and X_{BP26} is .4942. Hence, the removal of shared hidden units has actually made ID3 and BP less similar, rather than more similar as Hypothesis 2 suggests. The conclusion is that sharing in backpropagation is important to improving its performance, but it does not explain why ID3 and BP are performing differently.

3.3 Tests of Hypothesis 3: Statistics

We performed three experiments to test the third hypothesis.

In the first experiment, we took the outputs of the back-propagation network and thresholded them (values $> .5$ were mapped to 1, values $\leq .5$ were mapped to 0) before mapping to the nearest legal phoneme/stress pair. Table 6 presents the results for the 1000-word training set.

The results show that thresholding significantly

Table 5: Performance of 26 separate networks compared with a single network having 120 shared hidden units. Trained on 50-word training set, tested on 1000-word test set.

Method	Data set	Level of Aggregation (% correct)				
		Word	Letter	Phoneme	Stress	Bit (mean)
(a) ID3	TEST:	0.8	41.5	60.5	60.1	92.6
(b) BP 26 separate nets	TRAIN:	82.0	97.6	98.4	99.2	99.9
	TEST:	1.6	45.0	56.6	71.1	92.9
(c) BP 120 hidden units	TRAIN:	82.0	97.4	98.2	99.2	99.9
	TEST:	1.8	48.4	59.4	72.9	93.5
Difference (b)-(c)	TRAIN:	0.0	+0.2	+0.2	0.0	0.0
	TEST:	-0.2	-3.4***	-2.8**	-1.8*	-0.6
Difference (a)-(c)	TEST:	-1.0	-6.9	+1.1	-12.8	-0.9

Table 6: Performance of backpropagation with thresholded output values. Trained on 1000-word training set. Tested on 1000-word test set.

Method	Data set	Level of Aggregation (% correct)				
		Word	Letter	Phoneme	Stress	Bit (mean)
(a) ID3 (legal)	TEST:	9.6	65.6	78.7	77.2	96.1
(b) BP (legal)	TEST:	14.7	70.9	81.1	81.4	96.6
(c) BP (thresholded)	TEST:	12.1	67.9	78.6	80.3	96.6
Difference (c)-(b)	TEST:	-2.6***	-3.0***	-2.5***	-1.1*	0.0

drops the performance of back-propagation. Indeed, at the phoneme level, the decrease is enough to push BP below ID3. However, at the other levels of aggregation, BP still out-performs ID3. Nevertheless, the results support the hypothesis that the continuous outputs of the neural network aid the performance of BP. A comparison of correlation coefficients confirms this. The correlation between X_{ID3} and $X_{BPthresh}$ is .5598 (as compared with .5508 for X_{BP}).

While this experiment demonstrates the importance of continuous outputs, it does not tell us what kind of information is being captured by these continuous outputs nor does it reveal anything about the role of continuous weights inside the network. For this, we must turn to the other two experiments.

In the second experiment, we modified the method used to map a computed 26-bit string into one of the 324 strings representing legal phoneme/stress pairs. Instead of considering all possible legal phoneme/stress pairs, we restricted attention to those phoneme/stress pairs that had been observed in the training data. Specifically, we constructed a list of every phoneme/stress pair that appears in the training set (along with its frequency of occurrence). During testing, the 26-bit vector produced either by ID3 or BP is mapped to the closest phoneme/stress pair appearing in this list. Ties are broken in favor of the most frequent phoneme/stress pair. We call this the "observed" decoding method, because it is sensitive to the phoneme/stress pairs (and frequencies) observed in the training set.

Table 7 presents the results for the 1000-word training set and compares them to the previous technique ("legal") that decoded to the nearest legal phoneme/stress pair. The key point to notice is that this decoding method leaves the performance of BP virtually unchanged while it substantially improves the performance of ID3. Indeed, it eliminates a substantial part of the difference between ID3 and BP. Mooney et al. (1989), in their comparative study of ID3 and BP on this same task, employed a version of this decoding technique (without the tie-breaking by frequency), and obtained very similar results when training on a set of the 808 words in the dictionary that occur most frequently in English text.

An examination of the correlation coefficients shows that "observed" decoding increases the similarity between ID3 and BP. The correlation between $X_{ID3observed}$ and $X_{BPobserved}$ is .5705 (as compared with .5508 for "legal" decoding). Furthermore, "observed" decoding is almost always monotonically better (i.e., windows incorrectly classified by "legal" decoding become correctly classified by "observed" decoding, but not vice versa).

From these results, we can conclude that BP was already capturing most of the information about the frequency of occurrence of phoneme/stress pairs, but that ID3 was not capturing nearly as much. Hence, this experiment strongly supports Hypothesis 3.

The final experiment concerning Hypothesis 3 focused on extracting additional statistical information from the training set. We were motivated by Klatt's (1987) view that ultimately letter-to-phoneme rules will need to identify and exploit morphemes (i.e., commonly-occurring letter sequences appearing within words). Therefore, we analyzed the training data to find all letter sequences of length 1, 2, 3, 4, and 5, and retained the 200 most-frequently-occurring sequences of each length. For each retained letter sequence, we formed a list of all phoneme/stress strings to which

Table 7: Effect of "observed" decoding on learning performance.

Method	Data set	Level of Aggregation (% correct)				
		Word	Letter	Phoneme	Stress	Bit (mean)
(a) ID3 (legal)	TEST:	9.6	65.6	78.7	77.2	96.1
(b) BP (legal)	TEST:	14.7***	70.9***	81.1***	81.4***	96.6
(c) ID3 (observed)	TEST:	13.0	70.1	81.5	79.2	96.4
(d) BP (observed)	TEST:	14.9***	71.6*	81.8	81.4***	96.7
ID3 Improvement: (c)-(a)	TEST:	3.4***	4.5***	2.8***	2.0**	0.3
BP Improvement: (d)-(b)	TEST:	0.2	0.9	0.7	0.0	0.1

that sequence is mapped in the training set (and their frequencies). For example, here are the five pronunciations of the letter sequence "ATION" in the training set (Format is (⟨*phonemestring*⟩ ⟨*stressstring*⟩ ⟨*frequency*⟩)).

```
(("eS-xn" "1>0<<" 22)
 ("@S-xn" "1<0<<" 1)
 ("eS-xn" "2>0<<" 1)
 ("@S-xn" "2<0>>" 1)
 ("@S-xn" "1<0>>" 1))
```

During decoding, each word is scanned (from left to right) to see if it contains one of the "top 200" letter sequences of length k (varying k from 5 down to 1). If a word contains such a sequence, it is mapped and decoded as follows. First, each of the k windows in the sequence is evaluated and the results concatenated to obtain a bit string of length $k \cdot 26$. Then, this bit string is mapped to the nearest of the bit strings observed for this sequence in the training set (ties are broken in favor of the more-frequently occurring bit string). After decoding a block, control skips to the end of the matched k-letter sequence and resumes scanning for another "top 200" letter sequence of length k. After this scan is complete, the parts of the word that have not yet been matched are re-scanned to look for blocks of length $k-1$. We call this technique "block" decoding.

Table 8 shows the performance results on the 1000-word test set. Block decoding significantly improves both ID3 and BP, but again, ID3 is improved much more (especially below the word level). Furthermore, the correlation coefficient between $X_{ID3block}$ and $X_{BPblock}$ is .6747, which is a substantial increase compared to .5508 for legal decoding. Hence, block decoding also makes the performance of ID3 and BP much more similar.

Curiously, these summary numbers hide substantial shifts in performance caused by block decoding. To demonstrate this, consider that there is only a .7153 correlation between $X_{ID3legal}$ and $X_{ID3block}$. This reflects the fact that while "block" decoding gains 736 windows previously misclassified by "legal" decoding, it also loses 181 windows that were previously correctly classified by "legal" decoding. Similarly, there is only a .7746 correlation between $X_{BPlegal}$ and $X_{BPblock}$ (reflecting a gain of 433 and a loss of 226 windows).

The conclusion we draw is that block decoding further reduces the differences between ID3 and BP, and hence that this experiment also supports Hypothesis 3. The experiment suggests that the block decoding technique is a useful adjunct to any learning algorithm applied in this domain. It also suggests that the performance of block decoding could be improved if some way could be found to avoid losing windows that were correctly classified without block decoding. One technique we are exploring is to combine the constraints of blocks that overlap.

4 Discussion

The results shown in previous sections demonstrate that ID3 and BP, while they attain similar levels of performance, still do not cover the *same* set of testing examples. In particular, an analysis of the 7,242 7-letter windows in the test set reveals that there are 917 windows that are incorrectly classified by one of the algorithms and correctly classified by the other. This suggests that an inductive learning algorithm should be able to label correctly all of these 917 windows. This would yield a performance of 79.9% at the letter level, which would be quite good.

Other directions for improving these algorithms include (a) design of better error-correcting codes, (b) block decoding using overlapping blocks, (c) direct analysis of the training set to identify morphemes.

Klatt (1987) points out three properties of the domain that present special challenges to inductive learning methods:

> (1) the considerable extent of letter context that can influence stress patterns in a long word (and hence affect vowel quality in words like "photograph/photography"), (2) the confusion caused by some letter pairs, like CH, which function as a single letter in a deep sense, and thus misalign any relevant letters occurring further from the vowel, and (3) the difficulty of dealing with compound words (such as "houseboat" with its silent "e"), i.e., compounds act as if a space were hidden between two of the letters inside the word.

Long-distance interactions pose a difficult problem for BP, since capturing them presumably requires a very wide window. This in turn requires a very large network with many weights, and these will be much more difficult and time-consuming to train. ID3 scales

Table 8: Effect of "block" decoding on learning performance.

Method	Data set	Level of Aggregation (% correct)				
		Word	Letter	Phoneme	Stress	Bit (mean)
(a) ID3 (legal)	TEST:	9.6	65.6	78.7	77.2	96.1
(b) BP (legal)	TEST:	14.7***	70.9***	81.1***	81.4***	96.6
(c) ID3 (block)	TEST:	17.5	73.2	83.3	80.4	96.7
(d) BP (block)	TEST:	19.9***	73.8	83.9	81.5*	96.7
ID3 Improvement: (c)-(a)	TEST:	7.9***	7.6***	5.1***	3.2***	0.6*
BP Improvement: (d)-(b)	TEST:	5.2***	2.9***	2.8***	0.1	0.1

very well as the number of irrelevant features grows, so it should be able to handle much wider windows without problems.

General solutions to the other two problems mentioned by Klatt appear to be quite challenging. Indeed, machine learning techniques have some distance to go before they match the performance of the best human-engineered rule sets. Klatt cites Bernstein and Pisoni (1980) as measuring the performance of their rule system to be 97% at the phoneme level. By comparison, ID3 trained on 19,003 words and tested using "block" decoding is 91.5% correct at the phoneme level (i.e., an error rate nearly three times as bad).

5 Conclusions

The relative performance of ID3 and Backpropagation on the text-to-speech task depends on the decoding technique employed to convert the 26 bits of the Sejnowski/Rosenberg code into phoneme/stress pairs. Decoding to the nearest legal phoneme/stress pair (the most obvious approach) reveals a substantial difference in the performance of the two algorithms.

Experiments investigated three hypotheses concerning the cause of this performance difference.

The first hypothesis—that ID3 was overfitting the training data—was shown to be incorrect. Three techniques that avoid overfitting were applied, and none of them improved ID3's performance.

The second hypothesis—that the ability of back propagation to share hidden units was a factor—was shown to be only partially correct. Sharing of hidden units does improve the classification performance of backpropagation and—perhaps more importantly—the learning performance of the gradient descent search. However, an analysis of the kinds of errors made by ID3 and backpropagation (with or without shared hidden units) demonstrated that these were different kinds of errors. Hence, eliminating shared hidden units does not produce an algorithm that behaves like ID3.

The third hypothesis—that backpropagation was capturing statistical information by some mechanism (perhaps the continuous output activations)—was demonstrated to be the primary difference between ID3 and BP. By adding the "block" decoding technique to ID3, the level of performance of the two algorithms in classifying test cases becomes statistically indistinguishable (at the letter and phoneme levels). Consequently, in tasks similar to the text-to-speech learning task, ID3 with block decoding is clearly the algorithm of choice—particularly for initial exploratory studies, where its speed is a tremendous advantage.

6 Acknowledgements

The authors thank Terry Sejnowski for providing the Nettalk phonemic dictionary, without which this work would have been impossible. Correspondence with Jude Shavlik, Ray Mooney, and Geoffrey Towell helped clarify the possible kinds of decoding strategies. This research was supported by NSF grant numbers CCR-87-16748 and IRI-86-57316 (Presidential Young Investigator Award) with matching support from SUN Microsystems.

7 References

Bernstein, J., and Pisoni, D. B., (1980). Unlimited text-to-speech system: description and evaluation of a microprocessor-based device. *Proceedings of the Int. conf. Acoust. Speech Signal Process, ICASSP-80*, 576–579.

Klatt, D. (1987). Review of text-to-speech conversion for English. *J. Acoust. Soc. Am., 82*, (3), 737–793.

McClelland, J. L., and Rumelhart, D. E. (1988). *Explorations in Parallel Distributed Processing*, Cambridge, MA: MIT Press.

Mingers, J. (1989). An empirical comparison of pruning methods for decision tree induction. *Machine Learning, 4* (2), 227–243.

Mooney, R., Shavlik, J., Towell, G., and Gove, A. (1989) An experimental comparison of symbolic and connectionist learning algorithms. *IJCAI-89: Eleventh International Joint Conference on Artificial Intelligence*. 775–80.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess endgames, in Michalski, R. S., Carbonell, J., and Mitchell, T. M., (eds.), *Machine learning: An artificial intelligence approach, Vol. I*, Palo Alto: Tioga Press. 463–482.

Quinlan, J. R. (1986a). The effect of noise on concept learning. In Michalski, R. S., Carbonell, J., and

Mitchell, T. M., (eds.), *Machine learning, Vol. II*, Palo Alto: Tioga Press. 149–166.

Quinlan, J. R. (1986b). Induction of Decision Trees, *Machine Learning,1* (1), 81–106.

Quinlan, J. R., (1987). Simplifying decision trees. *International Journal of Man-Machine Studies, 27*, 221–234.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E., and McClelland, J. L., (eds.) *Parallel Distributed Processing*, Vol 1. 318–362.

Sejnowski, T. J., and Rosenberg, C. R. (1987). Parallel networks that learn to pronouce English text. *Complex Systems, 1*, 145–168.

Learning from Data with Bounded Inconsistency

Haym Hirsh
Computer Science Department
Rutgers University
New Brunswick, NJ 08903

Abstract

This paper presents an approach to concept learning from inconsistent data that foregoes a solution to the full-blown problem and instead considers a subcase, called bounded inconsistency. Data are said to have bounded inconsistency when some small perturbation to the description of any bad instance will result in a good instance. The key idea to learning in the presence of bounded inconsistency is to not only consider concept definitions that correctly classify all the training data, but also those that miss some of the data by only a small amount. The approach is implemented using a generalization of Mitchell's version-space approach to concept learning.

1 Introduction

The problem of inductive concept learning—forming general rules from specific cases—has been well-studied in machine learning and artificial intelligence. In the simplified case that is often studied the concept learning problem is to find some general description in a concept description language that covers all given positive examples of an unknown concept, and includes no negative examples of the concept. However, in real-world applications data are often subject to error, and there may be no concept that correctly classifies all the data. When data are inconsistent, the learner will be unable to find a description classifying all instances correctly. In general, learning systems must generate reasonable results even when there is no concept consistent with all the data.

Much of the past work on learning from inconsistent data forms concept definitions that perform well but not perfectly on the data, viewing those instances not covered as anomalous (*e.g.*, [Michalski and Larson, 1978; Quinlan, 1986]). Instances are effectively thrown away, even if they are merely slightly errant. The approach taken here to learning from inconsistent data is to forego a solution to the full problem, and instead solve a subcase of the problem for one particular class of inconsistency that can be exploited in learning. The underlying assumption for this class of inconsistency, called *bounded inconsistency*, is that some small perturbation to the description of any bad instance will result in a good instance. When this is true, a learning system can search through the space of concept definitions that correctly classify either the original data, or small perturbations of the data. The definition that does best can be taken as the result of learning.

This is the approach taken here, and is implemented using a generalization of Mitchell's [1978] version-space approach to concept learning. Mitchell defines a version space to be the set of all concept definitions in a prespecified language that correctly classify training data—the positive and negative examples of the unknown concept. The generalized approach [Hirsh, 1989b; 1990], called *incremental version-space merging*, removes the assumption that there is always some concept definition that correctly classifies all the given data.

The paper begins with a description of bounded inconsistency, the form of inconsistency considered by this paper. The paper continues with the general solution to this problem, followed by its implementation with incremental version-space merging. Experimental results are then presented, followed by an overview of related work and a general discussion. A formal analysis of how the quality of results is influenced by the amount of data used in learning concludes the paper. Further details are presented elsewhere [Hirsh, 1989b].

2 Bounded Inconsistency

This paper addresses the problem of learning from inconsistent data by solving a subcase of the problem called bounded inconsistency. The underlying assumption for this class of inconsistency is that some small perturbation to the description of any bad instance will result in a good instance. Whenever an instance is misclassified with respect to the desired final concept definition, some nearby instance description has the original instance's classification.

Figure 1 shows a simple way to view this. Concepts (such as C) divide the set of instances (I) into positives and negatives. I_1^+ is an example of a representative positive example. It is correctly classified with respect to the desired concept C. Similarly, I_2^- is a correctly classified representative negative example. I_3^+ however is incorrectly classified as positive even though the desired concept would label it negative. However, a neighboring instance description, I'^+_3, is near it across the border, and is correctly classified. Similarly for the incorrectly classified negative instance I_4^- and its neighbor I'^-_4. Roughly speaking, if misclassifica-

tions only occur near the desired concept's boundary, the data have bounded inconsistency.

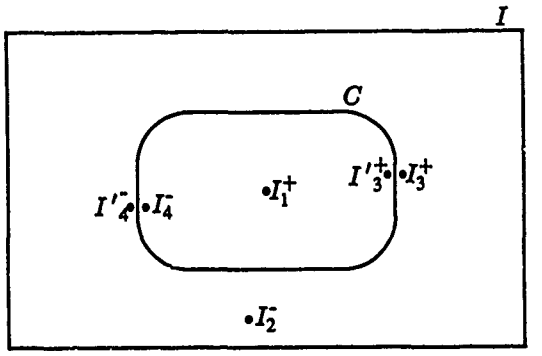


Figure 1: Pictorial Representation of Bounded Inconsistency.

For example, if instances are described by conjunctions of features whose values are determined by measuring devices in the real world, and the measuring devices are known to only be accurate within some tolerance, bounded inconsistency can occur. Consider an instance that is classified as positive, with a feature whose value is 5.0 (obtained by the real-world measuring device). If the tolerance on the measurement of the feature's value is 0.1, the instance could really have been one with feature value 4.9. If the "true" positive instance were 4.9, and the instance that really has value 5.0 would have been negative, a misclassification error has occurred. If the tolerance information is correct, for every incorrect instance there is a neighboring correct instance description, all of whose feature values are no more than the tolerance away from the original instance's value for that feature. This is an example of bounded inconsistency.

3 Approach

The approach taken to solve this problem of learning from data with bounded inconsistency is most easily described through the classic view of concept learning as search [Simon and Lea, 1974; Mitchell, 1978; 1982], namely that the goal of learning is to determine some concept definition out of a space of possible definitions as the desired result of learning. For consistent data this problem is simply one of finding a description that correctly classifies all the data. When data are inconsistent, however, there is no such definition. The approach taken here is to select concept definitions that correctly classify either all instances or, for those it does not, some other object in the instance language near the missed instance. Each instance is effectively "blurred," and the concept definitions to be considered are those that generate a correct classification for at least one instance in each "blur."

Note, however, that in general there will be many possible definitions that correctly classify some instance in each blur. The preferred result of learning is the description that requires the fewest instances to be blurred, that is, those concept definitions that correctly classify as much of the data as possible.[1]

4 Implementation

The method used to implement this approach is based on version spaces [Mitchell, 1978]. However, version spaces assume the data to be noise-free, and therefore some modification of the method is necessary.[2] The approach taken here is to use *incremental version-space merging* [Hirsh, 1989b; 1990], a generalization of version spaces [Mitchell, 1978] that removes its assumption of strict consistency with data. A version space is generalized to be any set of concept definitions in a concept description language representable by boundary sets.[3] The key observation is that concept learning can be viewed as the two-step process of specifying sets of relevant concept definitions and intersecting these sets. For each piece of information obtained—typically an instance and its classification—*incremental version-space merging* forms the version space containing all concept definitions that are potentially relevant given the information (determined as appropriate for the given learning task). The resulting version space is then intersected with the version space based on all past data. This intersection takes place in boundary-set form (using the *version-space merging algorithm* [Hirsh, 1989b; 1990]), and yields the boundary-set representation for a new version space that reflects all the past data plus the new information.

The general algorithm proceeds as follows:

1. Form the version space for the new piece of information.
2. Intersect this version space with the version space generated from past information.
3. Return to the first step for the next piece of information.

The initial version space contains all concept descriptions in the language, and is bounded by the S-set that contains the empty concept that says nothing is an example, and the G-set that contains the universal concept that says everything is an example.

Use of incremental version-space merging requires a specification of how the individual version spaces should be formed in the first step for each iteration. For example, using simple consistency with instances (*i.e.*, forming the version space of all concept definitions that correctly classify the current instance) results in an emulation of Mitchell's [1978] candidate-elimination algorithm. This and other examples are presented elsewhere [Hirsh, 1989b; 1989a; 1990], as are further details of the generalized version-space

[1] This is only one possible criterion for selecting a concept. Other criteria might consider the amount of "blurring" required to get an instance that matches the concept, and select the concept that minimizes the sum (or some other function) of the "blurs."

[2] However, Mitchell [1978] describes one possible way to extend his technique to learn from inconsistent data. This is discussed in Section 6.

[3] The boundary sets S and G contain the most specific and general concept definitions in the set. These bound the set of all concept definitions in the version space—the version space contains all concept definitions as or more general than some element in S and as or more specific than some element in G.

approach (including a discussion of the computational complexity of the method). The key idea in this work is to include in the instance version space concept definitions that are consistent with either the instance or some other term in the instance language near the given instance (where proximity is defined as appropriate for the given learning task). The version space of concept definitions to be considered for each instance will then be those concept definitions consistent with the instance or one of its neighbors within the region. The net effect is that all instances are "blurred," and version spaces reflect all instances within the blur.

The general technique can be viewed as follows:

Given:

- *Training Data*: Positive and negative examples of the concept to be identified.
- *Definition of Nearby*: A method that determines all instances near a given instance.
- *Concept Description Language*: A language in which the final concept definition must be expressed.

Determine:

- A set of concept definitions in the concept description language consistent with the data or nearby neighbors of the data.

The method proceeds as follows:

1. (a) Determine the set of instances near a given instance.
   (b) Form the version space of all concept definitions consistent with some instance in this set.
2. Intersect this version space with the version space generated from all past data.
3. Return to the first step for the next instance.

If an instance is positive the version space of all concept definitions consistent with some instance in the set of neighboring instances has as an S-set the set of most specific concept definitions that cover at least one of the instances, and as a G-set the universal concept that includes everything. If the single-representation trick holds (*i.e.*, for each instance there is a concept definition whose extension only contains that instance) [Dietterich *et al.*, 1982], the S-set contains the set of neighboring instances themselves. If an instance is negative the S-set contains the empty concept that includes nothing and the G-set contains all minimal specializations of the universal concept that excludes the instance or one of its neighbors.

4.1 Searching the Version Space

Ideally the result of this learning process would be a singleton version space containing the desired concept definition. However, if not given enough data the final version space will have more than one concept definition. This also happens if the definition of nearby is too generous, since it will allow too many concept definitions into the version space, and no set of instances will permit convergence to a single concept definition. The definition of nearby should be generous enough to guarantee that the desired concept definition is never thrown out by any instance, but not too generous to include too many things (or in the worst case, everything).

In addition, often it will take too long to wait for enough instances to lead to convergence to a single concept definition. As each instance throws away candidate concept definitions, the version space gets smaller and smaller. As the version space decreases in size, the probability that a randomly chosen instance will make a difference—will be able to remove candidate concept definitions—becomes smaller and smaller. The more data processed, the longer the wait for another useful instance. Therefore it will sometimes be desirable due to time considerations to use the small but nonsingleton version space (before converging to a single concept) to determine a usable result for learning.

Thus a situation can arise in which the final version space after processing data has multiple concept definitions. Not all of the remaining concept definitions are equal, though. Some may be more consistent with neighboring instances than given instances—for the concept definition to be consistent with ground data more instances require "blurring." An additional technique is therefore used to obtain a single final concept definition: When a nonsingleton final version space is generated, all candidate concept definitions in the version space are checked for their coverage of the raw, unperturbed data.[4] The concept definition with best coverage is then selected as the final generalization. This is computationally feasible as long as the version space is reasonably small in size.

5 Example

To demonstrate this technique Fisher's iris data [Fisher, 1936] is used. In addition to providing a test of the quality of the technique, it has been in use for 50 years, and thus permits a comparison to the results of other techniques.

5.1 Problem

The particular problem is that of classifying examples of different kinds of iris flowers into one of three species of irises: setosa, versicolor, and viginica. The goal is to learn three nonoverlapping concept definitions that cover the space of all irises; this requires a slight extension to the version-space approach, which is described in the next subsection. There are 150 instances, 50 for each class; instances are described using four features: sepal width, sepal length, petal width, and petal length. The units for all four are centimeters, measured to the nearest millimeter. For example, one example of setosa had sepal length 4.6cm, sepal width 3.1cm, petal length 1.5cm, and petal width 0.2cm.

The concept description language was chosen to be conjunctions of ranges of the form $a \leq x < b$ for each feature, where a and b are limited to multiples of 8 millimeters. An example of a legal concept description has [0.8cm $\leq$ petal length $<$ 2.4cm] and [petal width $<$ 1.6cm]. The range for defining neighboring instances was taken to be 3 millimeters for each feature—that is, as much as 3 millimeters can be added to or subtracted from each feature

[4] This, of course, requires retaining *all* data for this later check of coverage. An alternative strategy would be to retain only some subset of the data to lessen space requirements.

value for each instance (defining a range of size 6 millimeters centered on each feature value). There is no restriction on the number of features that may be blurred—anywhere from all to none may require blurring.

Note that although this means that each instance could be blurred to be any of an infinite number of instances within the range specified by the nearness metric (or if values are limited to the nearest millimeter, feature values can be blurred to any of a large number of nearby values), many of the instances are equivalent with respect to the concept description language. Two feature values, although different, can still fall in the same range imposed by the concept description language. Thus only a much smaller set of nearby instances need be considered and enumerated, one from each grouping of values imposed by the concept description language.

5.2 Method

The general approach of Section 4 was used to find rules. All neighboring instances for each instance are generated, by perturbing the instance in all ways possible—0.3 is added to and subtracted from each feature value, and the concept definitions consistent with each combination of potential feature values were formed. The union of all these concept definitions forms the version space for individual instances. For example, the positive instance of setosa given earlier (with sepal length 4.6cm, sepal width 3.1cm, petal length 1.5cm, and petal width 0.2cm), has four elements in its S-set. All have [2.8cm $\leq$ sepal width $<$ 3.6cm] and [0.0cm $\leq$ petal width $<$ 0.8cm]. They differ, however, on their restrictions on sepal length and petal length: the different concept definitions correspond to the four different combinations obtainable by choosing one of [4.0cm $\leq$ sepal length $<$ 4.8cm] and [4.8cm $\leq$ sepal length $<$ 5.6cm], and one of [0.8cm $\leq$ petal length $<$ 1.6cm] and [1.6cm $\leq$ petal length $<$ 2.4cm]. The G-set for the instance contains the universal concept that includes everything as positive.

The goal of learning is to form three disjoint concept definitions that cover the space of instances, and this requires two extensions to the technique described above. The first exploits the fact that the learned concept definitions must not overlap. The simple approach would be to take the 100 examples of two of the classes as negative examples for the third class. However, not only must the concept definitions for the third class exclude these instances, they must exclude all instances included by the final concept definition for each of the other two classes. It is not known what the final definitions will be, but it is known that they must be more general than some element of the S-set for its class. That is, whatever the concept definition, it must at least include all instances covered by some most specific concept definition generated from the positive data for that class. In the iris domain the S-set for each class after processing the positive data for that class was always singleton, so the final concept definition for each class must include all examples included by the final S-set element. Therefore, rather than taking the 50 examples of each class as negative data for the other two classes, first only positive data are processed for each class, and the resulting generalization in the S-set is taken as a generalized negative instance for the other two classes, replacing the use of the 100 negative instances with two generalizations that include the negative instances plus additional instances that must also be excluded. Thus, for example, all positive data are processed for the setosa class, and the result in the S-set is taken as a single generalized negative instance for versicolor and viginica. It summarizes the setosa data, as well as additional instances that will also be included by the final definition for setosa.

The second extension is that, since the three concept definitions that are formed must cover the space, many of the more specific definitions for each class can be removed, since no combination of definitions in the version spaces for the other two classes will cover the space of irises. Instead of applying the technique for searching the version space to find the best concept definition (Section 4.1), only the subset of the version space that could lead to class definitions that cover the space of irises is considered. The search then takes place in the cross products of the three much smaller version spaces. Furthermore, selection of a hypothesis in one space allows using it as a generalized negative instance for the other version spaces, so not all triples of concept definitions from the three version spaces need be considered.

5.3 Results

Since there is only a fixed amount of data, the learning technique was evaluated by dividing the data into 10 sets of 15 instances, five from each of the three classes. Learning took place by processing nine of the 10 data sets combined and testing on the tenth data set. This was done for each possible group of nine data sets, and the resulting classification rates were averaged across all 10 runs. A typical final result for a run is a rule set that classifies irises with [petal length $<$ 2.4cm] as setosa, irises with [petal length $\geq$ 2.4cm] and [petal width $<$ 1.6cm] as versicolor, and irises with [petal length $\geq$ 2.4cm] and [petal width $\geq$ 1.6cm] as viginica.

The average overall classification rate on the test data was 97%—on average 97% of the test cases were placed in the proper class. For the setosa class alone the rate was 100%, as the class is separable from the other two. For versicolor alone the rate was 93%, and for viginica 94%. These rates are comparable to those obtained with other techniques: for example, Dasarathy's pattern-recognition approach [Dasarathy, 1980] obtained 95% accuracy (100% for setosa, 98% for versicolor, and 86% for viginica); Aha and Kibler's noise-tolerant nearest-neighbor method NTgrowth [Aha and Kibler, 1989], also obtained 95% accuracy (100% for setosa, 94% for versicolor, and 91% for viginica); and C4 [Quinlan, 1987], Quinlan's noise-tolerant version of ID3, obtained 94% accuracy (100% for setosa, and 91% for versicolor and viginica).[5] The results are summarized in Table 1. NTgrowth and Dasarathy's method are both instance-based; C4 is the only other learning method that can be said to use a concept description language. For example, on one run it generated the decision tree that defines irises with [petal length $<$ 2.5cm] as setosa, irises with [petal length $\geq$ 2.5cm] and [petal width $<$ 1.8cm] as versicolor, and those remaining ([petal length $\geq$ 2.5cm] and [petal width $\geq$ 1.8cm]) as viginica. Note that, unlike this

[5] These latter two results are due to Aha (personal communication).

work, C4 decides for itself where attribute values should be divided.

Learning Algorithm	Setosa	Viginica	Versicolor	Overall
IVSM	100%	93.33%	94.00%	96.67%
Dasarathy	100%	98%	86%	94.67%
NTgrowth	100%	93.50%	91.13%	94.77%
C4	100%	91.07%	90.61%	94.00%

Table 1: Predictive Accuracy of Learning Systems.

6 Comparison to Related Work

This paper describes one approach to learning from inconsistent data by only addressing the subcase of data with bounded inconsistency. Drastal, Meunuer, and Raatz [Drastal *et al.*, 1989] have proposed a related method that works in cases where only positive data have bounded inconsistency. Their approach is to overfit the inconsistent data, using a learning technique capable of forming multiple disjuncts, some of which only exist to cover anomalous instances. After learning, they remove disjuncts that only cover instances that can be perturbed to fit under one of the other disjuncts, in effect removing the disjunctions that only exist to cover the anomalous data. One benefit of their technique is that it is applied after learning, focusing on only those instances covered by small disjuncts, whereas here all instances must be viewed as potentially anomalous. However, they make the stronger assumption that all such inconsistent data fall into small disjuncts. They furthermore only handle positive data.

As mentioned earlier, Mitchell [1978] presented an alternative approach to learning from inconsistent data with version spaces. The key idea was to maintain in parallel version spaces for various subsets of the data. When no concept definition is consistent with all data, Mitchell's approach considers those concept definitions consistent with all but one instance. As more inconsistency is detected, the system uses version spaces based on smaller and smaller subsets of the data, which the system has been maintaining during learning. The number of boundary sets that need be maintained by this process is linear in the total number of instances to be processed (in the worst case). This is still unacceptably costly. In the iris domain, assuming that at most 10% of the data should be discounted, this would have required updating 30 boundary sets for each instance. Even using the less reasonable assumption that only 4% of the data need be ignored will result in an order of magnitude slow down. Furthermore, Mitchell's approach requires knowing the absolute maximum number of incorrectly classified instances, in contrast to allowing unlimited number of errors as done here (replacing it with a bound on the distance any instance may be from a correct instance). Finally, the boundary sets for Mitchell's approach are much larger than for the noise-free case, since Mitchell modifies the candidate-elimination algorithm S-set updating method to ignore negative data, and similarly positive data are ignored by the modified G-set updating method (this allows the use of a linear number of boundary sets by keeping the boundary sets for multiple version spaces in a single boundary set).

A significant distinction can be made between this work and Mitchell's approach, as well as much other work in learning from inconsistent data (*e.g.* [Michalski and Larson, 1978; Quinlan, 1986]. These approaches form concept definitions that perform well but not perfectly on the data, viewing those instances not covered as anomalous. Instances are effectively thrown away, whereas here every instance is viewed as providing useful information, and the final concept definition must be consistent with the instance or one of its neighbors. The approach presented here works on data with bounded inconsistency. Other approaches handle a wider range of inconsistency, but cannot utilize any instances that are just a small ways off from correct; they instead throw out such instances as nonhelpful. Furthermore, unlike other approaches that degenerate as more inconsistency is imposed on the data, the incremental version-space merging approach described here still succeeds even when all of the data are subject to bounded inconsistency.

To further demonstrate this point a series of runs of the learning method were done on a simple, artificial domain. There are three attributes that take on real values in the range of 0 to 9. The concept description language partitions attributes into three regions: greater than or equal to 0 and less than 3, greater than or equal to 3 and less than 6, and greater than or equal 6 and less than or equal to 9. A concept definition is a conjunction of such ranges over the various attributes. A single preselected concept definition serves as the target of learning.

Data were created by randomly generating some value for each attribute in its legal range (0 to 9). This instance was then classified according to the preselected target concept definition for learning. The identity of each instance is then perturbed by up to one unit—a random number between -1 and 1 is added to the given value of each attribute. This new instance is given the classification of the instance on which it is based. Training data generated in this manner have bounded inconsistency, since any incorrect instance is never more than 1 away from a correct instance on any attribute.

The test runs perturb different percentages of the data, to test the sensitivity of the approach to this factor. The definition of "nearby" used by the learning method defines one instance to be near another if the value of each attribute of the first are within 1 unit of the corresponding value for the second (i.e., the appropriate definition of nearby was selected). All attribute values for a single instance may be perturbed. 80 randomly generated instances were used.

Table 2 summarizes the results of the test. The amount of data that was perturbed was allowed to vary from 0% (no data perturbed—data are consistent) to 100% (all data perturbed). In all cases learning used a definition of "nearby" that added 1 to and subtracted 1 from the value of each attribute. The result of the experiment was that, in all cases, incremental version-space merging (with the additional step of selecting the best classifier from the version space) converged to the target concept definition that was used to generate the data. Unlike most other learning algorithms that degenerate as more noise is introduced to the data, the

technique was able to correctly learn the desired concept definition even when all data are perturbed within known bounds. One way to interpret these results is that the approach described here provides a way to use the knowledge that bounded inconsistency exists, which permits successful learning even when all the data are incorrect (within the known bounds).

%	Correct?
0	Yes
20	Yes
40	Yes
60	Yes
80	Yes
100	Yes

Table 2: Correct Concept Identification for Different Amounts of Inconsistency.

7 Discussion

The general approach described here is to consider concept definitions consistent with the instance or some neighbor of the instance. The technique requires a method for generating all instances near a given instance, but it does not constrain a priori the particular definition of "nearby." For example, in tree-structured description languages one such definition would be that two feature values are in the same subtree: rhombus and square might be close whereas rhombus and oval might not.

However, the approach described here is extremely sensitive to both the concept description language and the definition of "nearby." For a fixed language, if the notion of nearby is too small, the version space will collapse with no consistent concept; if it is too large, each instance will have many neighbors, and instance boundary sets will be quite large, which makes the approach computationally infeasible. Furthermore, the final version space will likely be too big to search for a best concept after processing the data. Similarly, for a fixed definition of nearby, if the concept description language is too coarse, instances will have no neighbors, whereas if the language is too fine, then instances will have too many neighbors. The choice of language and definition of nearby affects the size of version spaces and the convergence rate for learning (how many instances are required to converge, if it is even possible).

The ideal situation for this approach would be when the definition of nearby is given or otherwise known for the particular domain, as well as when the desired language for concepts is provided. However, it is often the case that one or both are not known, as was the case for the iris domain of the previous section, which required a few iterations before a successful concept description language and definition of nearby was found. The first description language chosen used intervals of size 4 millimeters, rather than 8 as was finally selected. The first definition of nearby considered instances with feature values within 2 millimeters of given feature values as "nearby," but the version space collapsed—no concept definition remained after processing the data—for the versicolor and viginica classes. However, a definition of nearby of 3 millimeters resulted in too many neighbors for each instance, making the process too computationally expensive (the program ran out of memory). Since measurements were only given to the nearest millimeter, there was no intermediate definition of nearby to try. Since adjusting the definition of nearby failed to work, the next step was to adjust the language, and 8 millimeter intervals were chosen. Fortunately the first attempt using the new language with a definition of nearby of 3 millimeters yielded nonempty version spaces of reasonable size. Criteria for selecting appropriate description languages and definitions of nearby is an area for future work.

To further demonstrate this issue a number of runs were made on the artificial learning task of the previous section. In these experiments the concept description language was fixed (using the same language as in the previous section) and the definition of nearby (the amount of inconsistency assumed present by the learning method) was varied. 100% of the attribute values were perturbed by up to 1 unit. This set of experiments explores how the definition of nearby affects the size of the final version space, as well as the sizes of the boundary sets for each instance version space. The computational complexity of incremental version-space merging is sensitive to boundary-set size [Hirsh, 1989b; 1990].

Nearby	Average $\|VS\|$	Average # Neighbors
0	0	1
1	1	3
2	4	6.48
3	38	12.91

Table 3: Version Space Size and Number of Neighbors for Different Definitions of "Nearby."

The results of these experiments are summarized in Tables 3 and 4. There are three attributes, and altogether there are 216 concept definitions in the language. The different rows correspond to different definitions of nearby—how much is added to or subtracted from each instance. This was varied from 0 to 3—the maximum distance apart the feature values of two instances can be so that the instances are still considered neighbors (column 1 in the tables). Note that the real amount of variance imposed on values when generating data was at most 1—no more than 1 was added to or subtracted from the randomly generated value for the feature. The second column of Table 3 summarizes the size of the final version space after all 80 instances have been processed. Note that, while learning is impossible here if the data are assumed consistent, and convergence is possible using the 80 instances if the definition of nearby adds or subtracts only 1 to each value (the actual value used in generating the inconsistent data), as the value for nearby increases to 2 and 3 the final version-space size increases. The third column presents the average number of neighbors for each instance. As would be expected, this number increases significantly (exponentially) as the amount by which values may be perturbed increases. The values are close to their expected values (obtainable by case enumeration) of 1.0, 3.0 (= $(13/9)^3$), 6.7 (= $(17/9)^3$), and 12.7 (= $(7/3)^3$).

Table 4 summarizes the average size of boundary sets for each instance. Since positive instances always have a singleton G-set, and similarly negative instances always have a singleton S-set, averages are also given for the S- and G-sets when excluding these cases. As expected, as the definition of nearby grows more generous, the various quantities increase.

	Average $\|S\|$	Average Pos $\|S\|$	Average $\|G\|$	Average Neg $\|G\|$
0	1	1	3.34	4.34
1	1.51	2.71	3.88	5.11
2	2.54	6.13	4.40	5.86
3	4.01	11.04	5.06	6.80

Table 4: Boundary Set Size for Different Definitions of "Nearby."

These results emphasize the need for a sufficiently generous, but not overly generous, definition of nearby. They furthermore suggest a method for automating the selection of an appropriate definition of nearby given a fixed concept description language. The method would begin by assuming consistency, then slowly increase the amount of inconsistency assumed to be present in the data (i.e., increase the generosity of the definition of nearby) until either a nonempty version space is generated or enough time has passed to believe that the approach is not computationally feasible on the given data with the given concept description language.

8 Formal Results

Recent theoretical work on concept learning (e.g., [Haussler, 1988]) has developed techniques for analyzing how the quality of results is influenced by the amount of data used in learning. This section gives such an analysis for learning from data with bounded inconsistency.

To do this a number of definitions are necessary. First, the function *Neighbors(x)* gives the set of examples in the instance description language that are near x (for the learning-task specific definition of nearby):

Definition 1:

$$Neighbors(x) = \{y \mid y \text{ is near } x\}.$$

Furthermore, since data may have bounded inconsistency, it is necessary to redefine what it means for a concept definition to be consistent with an instance. A concept is consistent with an example if it correctly classifies the example or one of its neighbors:

> **Definition 2:** An instance x is said to be consistent with a concept C (written $Consistent(x, C)$) if, when x is positive, $\exists p \in Neighbors(x)$ with $p \in C$, and when x is negative, $\exists n \in Neighbors(x)$ with $n \notin C$ (where $p \in C$ means C would have classified p as positive, and $n \notin C$ means C would have classified n as negative).

The definition of the error of a concept definition h with respect to a desired target concept C is now defined relative to this new notion of consistency—it is the probability that h and C disagree:

> **Definition 3.** $Error(h, C)$ = the probability that for a randomly chosen instance x classified as a positive or negative example of C, it is not the case that $Consistent(x, h)$.

With these definitions it is possible to map over Lemma 2.2 from Haussler's [1988] *A.I. Journal* paper:

> **Lemma 1:** The probability that some element of the version space generated from m examples of C has error greater than ϵ is less than $|H|e^{-\epsilon m}$, where $|H|$ is the number of expressions in the concept description language H used by incremental version-space merging.

Proof: Assume that some set of hypotheses $h_1, \ldots, h_k$ in the concept description language H have error greater than ϵ with respect to C. This means that the probability that an example of C is consistent with a particular h_i is less than $(1 - \epsilon)$. The probability that h_i is consistent with m independent examples of C is therefore less than $(1 - \epsilon)^m$. Finally, the probability that *some* $h_i \in h_1, \ldots, h_k$ is consistent with m instances is bounded by the sum of their individual probabilities, thus the probability that some h_i with error greater than ϵ (with respect to C) is consistent with m examples of C is less than $k(1 - \epsilon)^m$. Since $k \leq |H|$, and $(1 - \epsilon)^m \leq e^{-\epsilon m}$, the probability of getting some hypothesis with error greater than ϵ consistent with m independent examples of C is less than $|H|e^{-\epsilon m}$. □

A simple corollary of this Lemma says how many examples are necessary to guarantee with high probability that all concept definitions in the version space have low error:

> **Corollary 1:** The probability that all elements of the version space generated from at least
>
> $$\frac{ln(|H|) + ln(\frac{1}{\delta})}{\epsilon}$$
>
> examples of C will have error less than ϵ is $1 - \delta$.

Proof: Solving $\delta < |H|e^{-\epsilon m}$ for m gives the desired result. □

At first these results may appear somewhat surprising, since they give the same guarantees as Haussler, yet address learning from inconsistent data. The reason for this is that these results use a weaker definition of consistent, and hence use a weaker notion of error, than the traditional definition used by Haussler.

Finally, note that these results do not only apply to the version space approach presented here. Any learning method that generates concept descriptions "consistent" (in the sense of Definition 2) with m examples of some concept C will have these guarantees.

9 Summary

This paper has described an approach to learning from inconsistent data that focuses on a subcase of the problem, when data have bounded inconsistency. The key

idea to learning from such data is to find concept definitions that miss some of the data, but only by a small amount. This approach was implemented using a generalization of Mitchell's version-space approach to concept learning called incremental version-space merging. The central idea is to allow version spaces to contain concept definitions not consistent with the data, but instead consistent with neighboring data. This results in considering more concept definitions than the original version-space approach would ordinarily consider, decreasing the chance that the desired concept definition is removed due to an anomalous instance.

Acknowledgments

This paper summarizes part of the author's dissertation [Hirsh, 1989b], where more lengthy acknowledgments can be found. Among the people who have had an impact on the portion presented here are David Aha, Jim Blythe, Bruce Buchanan, William Cohen, Oren Etzioni, Rich Keller, Tom Mitchell, Paul Rosenbloom, Jeff Schlimmer, and members of the GRAIL and Logic groups at Stanford and the MLRG group at CMU. David Aha provided the iris data (originally entered by Mike Marshall at NASA Ames and now one of the many data sets in the U. C. Irvine Repository of Machine Learning Domains), and generated the results for NTgrowth and C4. William Cohen helped with the formal results in Section 8.

References

[Aha and Kibler, 1989] D. W. Aha and D. Kibler. Detecting and removing noisy instances from concept descriptions. Technical Report 88-12, University of California, Irvine, 1989.

[Dasarathy, 1980] B. V. Dasarathy. Nosing around the neighborhood: A new system structure and classification rule for recognition in partially exposed environments. *PAMI*, PAMI-2(1):67–71, January 1980.

[Dietterich *et al.*, 1982] T. G. Dietterich, B. London, K. Clarkson, and G. Dromey. Learning and inductive inference. In P. Cohen and E. A. Feigenbaum, editors, *The Handbook of Artificial Intelligence, Volume III*. William Kaufmann, Los Altos, CA, 1982.

[Drastal *et al.*, 1989] G. Drastal, R. Meunier, and S. Raatz. Error correction in constructive induction. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 81–83, Ithaca, New York, 1989.

[Fisher, 1936] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annual Eugenics*, 7:179–188, 1936. Also in *Contributions to Mathematical Statistics*, John Wiley & Sons, NY, 1950.

[Haussler, 1988] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 26(2):177–221, Sept. 1988.

[Hirsh, 1989a] H. Hirsh. Combining empirical and analytical learning with version spaces. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 29–33, Ithaca, New York, 1989.

[Hirsh, 1989b] H. Hirsh. *Incremental Version-Space Merging: A General Framework for Concept Learning*. PhD thesis, Stanford University, June 1989.

[Hirsh, 1990] H. Hirsh. Incremental version-space merging. In *Proceedings of the Seventh International Conference on Machine Learning*, Austin, Texas, June 1990.

[Michalski and Larson, 1978] R. S. Michalski and J. B. Larson. Selection of most representative training examples and incremental generation of vl1 hypotheses: The underlying methodology and description of programs esel and aq11. Report 867, University of Illinois, 1978.

[Mitchell, 1978] T. M. Mitchell. *Version Spaces: An Approach to Concept Learning*. PhD thesis, Stanford University, December 1978.

[Mitchell, 1982] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, March 1982.

[Quinlan, 1986] J. R. Quinlan. The effect of noise on concept learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Volume II*, pages 149–166. Morgan Kaufmann, Los Altos, CA, 1986.

[Quinlan, 1987] J. R. Quinlan. Decision trees as probabilistic classfiers. In *Proceedings of the Fourth International Machine Learning Workshop*, pages 31–37, Irvine, CA, June 1987.

[Simon and Lea, 1974] H. Simon and G. Lea. Problem solving and rule induction. In H. Simon, editor, *Models of Thought*. Yale University Press, 1974.

Conceptual Set Covering: Improving Fit-And-Split Algorithms

Carl M. Kadie[1]
Knowledge-Based Systems Group,
Department of Computer Science & Beckman Institute,
University of Illinois, Urbana, IL 61801

Abstract

Many learning systems implicitly use the fit-and-split learning method to create a comprehensive hypothesis from a set of partial hypotheses. At the core of the fit-and-split method is the assignment of examples to partial hypotheses. To date, however, this core has been neglected. This paper provides the first definition and model of the fit-and-split assignment problem. Extant systems perform assignment nearly arbitrarily, implicitly using, for example, greedy set covering. This paper also presents Conceptual Set Covering (CSC), a new assignment algorithm. An extensive empirical evaluation over a wide range of learning problems suggests that CSC can improve any fit-and-split learning system.

1 Introduction

One way to solve a complex learning problem is to decompose it into simpler learning problems. The *fit-and-split* learning method is perhaps the most direct way to do such a decomposition.

FIT-AND-SPLIT LEARNING METHOD
Input: Examples and learning problem called the partial learner
Procedure:
1) Fitting — Use the partial learner to find a set of partial hypotheses such that each example is covered by at least one partial hypothesis.
2) Splitting, part 1 — Assign each example to one of the partial hypotheses that covers it.
3) Splitting, part 2 —For each partial hypothesis, create a general decision rule that covers the assigned examples.
Output: A final hypothesis in the form of a decision ?? listsis de

```
if decision rule1
  then apply partial hypothesis1
else decision rule2
  then apply partial hypothesis2
...
else decision rulen
  then apply partial hypothesisn
```

The fit-and-split learning method is used by many machine learning systems of every type. For example, it is used by operator learners [Kadie, 1989; Shen and Simon, 1989], by automatic programmers [Summers, 1977], by discovery systems [Falkenhainer and Michalski, 1986], and by integrated empirical/explanation-based systems [Drastal et al., 1989]. All these systems concentrate on step one of the method (partial-hypothesis creation) or step three (decision-rule creation). Extant systems typically neglect step two, example assignment; they do the assignment without regard to the effect on the final hypothesis using, for example, the greedy-set-covering algorithm.

This paper describes a new assignment algorithm called Conceptual Set Covering (CSC). CSC tries to assign examples to partial hypotheses so as to minimize the total number of disjuncts in the decision rules of the final hypothesis. The result is a final hypothesis that is usually simpler and more accurate than that produced by alternative methods. Empirical evidence suggests that the adoption of CSC can improve any fit-and-split learner.

2 Fit-and-Split Learning Problem

This section illustrates the fit-and-split learning problem with an example, defines the learning problem, and then uses a learning-problem generator to specify a parameterized model of typical fit-and-split learning problems. Later the model will be used to motivate a heuristic method.

2.1 Example

Consider an example from the domain of empirical discovery. The target function (unknown to the learner) is:

$$\text{If } x \geq 1 \text{ then } \sin\left(\frac{\pi}{2}x\right)$$
$$\text{else if } x \leq -4 \text{ then } 1$$
$$\text{else } -x.$$

In addition, suppose that the learning program is given five example input/output pairs: $(-5,1), (-2,2), (-1,1), (2,0)$, and $(5,1)$. Figure 1a shows the target function and the examples. As the learner's first step, it calls the partial learner to create a set of partial hypotheses. Suppose the partial learner creates the functions shown in figure 1b.

1 Support was provided by the Fannie and John Hertz Foundation and ONR grant N00014-88-K124.

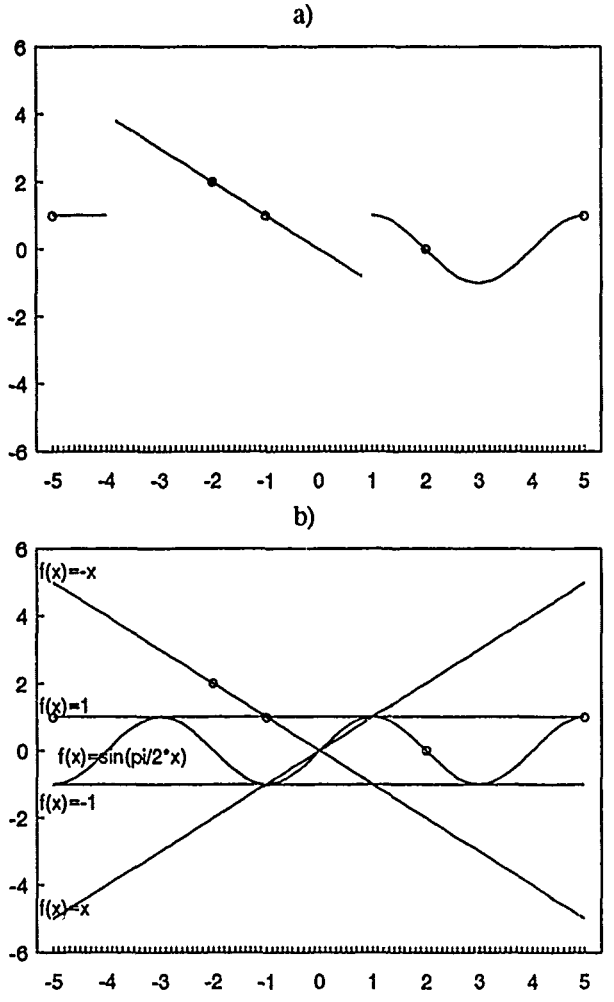


Figure 1: a) The Target Function and Example Points — From the example points the learner tries to create an accurate approximation of the function. b) The partial learner is used to create a set of partial hypotheses such that each example is covered by at least one partial hypothesis. Here, each example is an input/output pair and each partial hypothesis is a function. A partial hypothesis covers an example if it includes the point defined by the example.

None of these partial hypotheses cover all examples, so they must be combined in a decision list. The decision rules for the decision list are typically formed by a concept learner such as AQ, ID3, or PLS [Michalski and Chilausky, 1980; Quinlan, 1986; Rendell, 1986]. Such a concept learner takes as input a set of positive and negative examples. It returns a decision rule that covers (that is, returns true when applied to) each positive example but does not cover any negative example.

The assignment problem is thus two-fold. First, the assigner must decide where in the decision list each partial hypothesis will appear.[2] Second, for each partial hypothesis occurrence, the assigner must decide which examples are to be covered by that occurrence. The next section shows how the choices made by the assigner can affect the complexity (and thus the accuracy) of the final hypothesis.

2.2 Fit-and-Split-Assignment Problem

A geometric interpretation of the assignment problem will help illustrate how good assignments lead to good final hypotheses and how bad assignments lead to bad final hypotheses.

A decision list, such as the final hypothesis, can be plotted as an ordered set of orthogonal rectangles.[3] Each rectangle represents a disjunct within a decision rule of the decision list. The desire for a simple final hypothesis translates into a preference for the decision list plotted in figure 2b over the one plotted in figure 2a, because figure b uses fewer rectangles and, thus, fewer disjuncts.

The fit-and-split problem can be formalized in terms of the geometric interpretation:

FIT-AND-SPLIT PROBLEM
Given: A set of examples and a partial learner
Find: A set of covering partial hypotheses and a set of rectangles (disjuncts) such that:
- Every rectangle is labeled with a partial hypothesis.
- The set of rectangles is ordered. (The relative priority of rectangles with the same label is not important.)
- Every example must be inside at least one rectangle.
- Every example must be compatible with the first rectangle that it lies within. An example is said to be compatible with a rectangle if the example is covered by the rectangle's label.
- **The number of rectangles is as small as possible.**

2 A partial hypothesis can appear more than once in the decision list.

3 The term *rectangles*, as used here, includes hyper-rectangles and open-ended rectangles which extend to infinity along one or more axes.

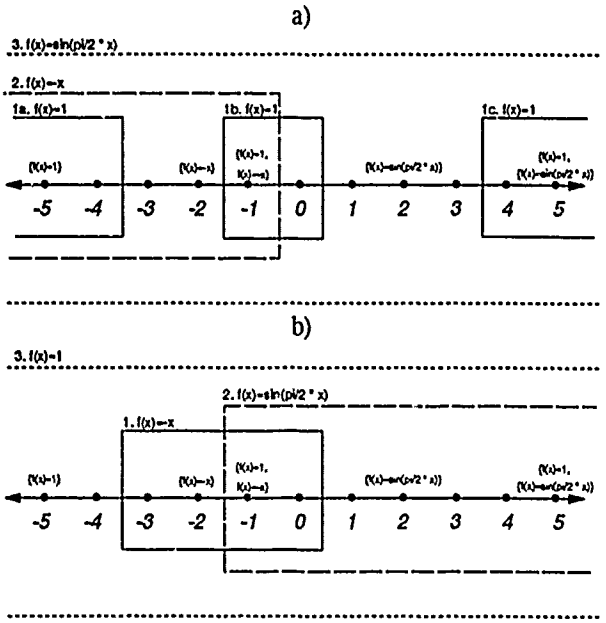


Figure 2: Geometric Interpretation — Examples are represented by labeled points. Disjuncts are represented with rectangles. Examples are labeled with the partial hypotheses that cover them. Rectangles are labeled with a single partial hypothesis and a number. Low-numbered rectangles have priority over high-numbered rectangles. Figure a shows the examples covered with five rectangles (and three partial hypotheses). Figure b shows the examples covered with three rectangles (and three partial hypotheses). Because figure b covers the examples with fewer rectangles, it is preferred.

Table 1: The Assignment Lists Corresponding to Figure 2.

a)

Partial Hypothesis	Assigned Examples
1. $f(x)=x$	$\{-5,-1,5\}$
2. $f(x)=-x$	$\{-2\}$
3. $f(x)=\sin\left(\frac{\pi}{2}x\right)$	$\{2\}$

b)

Partial Hypothesis	Assigned Examples
1. $f(x)=-x$	$\{-2,-1\}$
2. $f(x)=\sin\left(\frac{\pi}{2}x\right)$	$\{2,5\}$
3. $f(x)=1$	$\{-5\}$

Rather than looking directly for the ordered set of rectangles, a learner can look for an *assignment list*. Table 1 shows the assignment list corresponding to figure 2. An assignment list is an ordered set of two pairs. The first component of each pair is a partial hypothesis. The second component is the set of examples that are assigned to that partial hypothesis. An example, (x, y), is represented by its x component. Although not illustrated here, a partial hypothesis may appear in a list more than once (or not at all). Examples, however, appear exactly once. In addition, an example must be assigned to a partial hypothesis that covers it. Rectangles are formed from assignment lists according to the procedure listed in figure 3.

1. For each element, $<PARTIAL\_HYPO_i, x\_set_i>$ in the assignment table (in order):

1a. Let $POS = x\_set_i$. Let $NEG = \bigcup_{j=i+1}^{n} x\_set_j$, where n is the length of the assignment table.

1b. Use a concept learning program to create a set of rectangles REC_SET_i, that distinguishes POS from NEG.

2. Output

"If REC_SET_1 then $PARTIAL\_HYPO_1$
else if REC_SET_2 then $PARTIAL\_HYPO_2$
...
else if REC_SET_n then $PARTIAL\_HYPO_n$"

Figure 3: The Assignment-List-to-Decision-List Algorithm — For each row of the table, the algorithm produces a decision rule that distinguishes the examples on the row from the examples on subsequent rows.

Now the fit-and-split-assignment problem, a subproblem of the fit-and-split problem, can be defined:

FIT-AND-SPLIT-ASSIGNMENT PROBLEM
Given: A set of examples and a covering set of partial hypotheses.
Find: An assignment list such that the decision list produced by the assignment-list-to-decision-list algorithm will contain as few rectangles as possible.

In the one-dimensional case the problem does not look difficult. The example space, however, can be n-dimensional and non-Euclidean. In general the problem is $\mathcal{NP}$-hard (because any set-covering problem [Garey and Johnson, 1979] can be reduced to a fit-and-split assignment problem with an example space consisting of one nominal dimension).

Because the problem is $\mathcal{NP}$-hard, it is natural to look for heuristic methods with good performance on typical cases, but first, typical fit-and-split-assignment problems must be characterized. This will be done by modeling fit-and-split-assignment problems with a parameterized problem generator.

2.3 Model of Fit-and-Split-Assignment Problems

The first step toward describing a model of assignment problems is to introduce some new terms. In the example problem, three examples are *unambiguous*, that is, each is covered by exactly one partial hypothesis, called its *true partial hypothesis*. Unambiguous examples constrain assignment algorithms because each unambiguous example must be assigned to the example's true partial hypothesis. *Ambiguous* example are those that are covered by two or more partial hypotheses: one true partial hypothesis and one or more *coincidental partial hypotheses*. A look back at the target function of the example problem (figure 1a), shows that example $(-1, 1)$ is covered by its true partial hypothesis, $f(x) = -x$, and one coincidental partial hypothesis, $f(x) = 1$. Likewise, example $(5, 1)$ is truly covered by $f(x) = \sin\left(\frac{\pi}{2}x\right)$ and coincidentally covered by $f(x) = 1$. In general, every example will be assumed to have one true partial hypothesis, and zero or more coincidental partial hypotheses. The success of an assignment algorithm will depend in part on how well it can distinguish true partial hypotheses from coincidental partial hypotheses.

With these terms in mind, a typical fit-and-split-assignment problem can be characterized by describing a problem generator. The generator is listed in figure 4.

2.4 Related Learning Problems

The fit-and-split-assignment problem is a specialization of the overlapping-concept-learning problem [Michalski, 1983; Cohen and Feigenbaum, 1982], where a *class* corresponds to a *partial hypothesis*. It differs from standard overlapping concept learning in two ways. First, examples are labeled with *every* class (that is, partial hypothesis) that covers them. In the standard problem, examples are labeled with only one covering class at a time. Second, every example is covered by one true class and zero or more coincidental classes. In other words, examples are always correctly labeled with their true class, but because of noise/coincidence, they may also be labeled with zero or more coincidental classes. Unlike other noise models, coincidence is consistent, that is, if an example comes up again, it will be labeled the same way. In the standard problem, overlap is caused by examples having two or more true classes.

1. Start with an example space. In general, any example space is permissible. (For the tests of section 4, the example space is $[1,10] \times [0.0, 10.0] \times \{red, green, blue\} \times [1,20]$.)
2. Randomly partition the space into regions. The space is split by repeatedly dividing the space with a hyperplane. The hyperplane is defined with a randomly chosen value on a randomly chosen axis. The random choices are made according to uniform distributions. The number of splits to be made is specified by the product of two parameters, *number of disjuncts per true partial hypotheses* and *number of true partial hypotheses*.
3. Partial hypotheses are randomly selected from a finite set of possible true partial hypotheses, *PT*. The selection is done according to the uniform distribution. A parameter called *number of true partial hypotheses* specifies the size of *PT*.
4. Example points are selected from the space by randomly selecting a value on each axis. Values are chosen according to the uniform distribution.

5a. If the point has been generated before, the partial hypothesis that covered it before covers it now.

5b. If the point is new, it is always covered by the true partial hypothesis of the region from which it comes and may be covered by coincidental partial hypotheses. Coincidental partial hypotheses come from *PC*, a set formed by randomly adding between 0 and $|PT|^2 - 1$ new partial hypotheses to the set *PT*. The probability that a partial hypothesis in *PC* will cover an example is a parameter called *chance of coincidence*.

Figure 4: Assignment Problem Generator — This generator provides a parameterized model of typical assignment learning problems. The parameters are *number of true partial hypotheses*, *number of disjuncts per true partial hypotheses*, and *chance of coincidence*.

3 Conceptual-Set-Covering Algorithm

The most straightforward way to try to solve a fit-and-split assignment problem is with the greedy-set-covering algorithm shown in figure 5.

1. Let $i = 1$.
2. Find, *BEST_PARTIAL_HYPO*, the partial hypothesis that covers the most examples.

2a. Let $PARTIAL\_HYPO_i = BEST\_PARTIAL\_HYPO$. Let x_set_i be the x components of all the examples that *BEST_PARTIAL_HYPO* covers. Remove all the covered examples from consideration.

2b. Increment i and repeat at step 1 until all examples are removed.

3. Return the assignment list

Figure 5: The Greedy-Set-Covering Algorithm

The problem with the greedy-set-covering algorithm is that it does not go far enough. It tries to minimize the number of decision rules in the final hypothesis, but does not try to minimize the number of disjuncts within those decision rules.

When research on the clustering problem faced a similar situation, *conceptual-clustering* algorithms were developed. These are algorithms that look for a solution that can be expressed simply [Pitt and Reinke, 1988; Stepp and Michalski, 1989]. In a similar manner, the Conceptual-Set-Covering algorithm (CSC) tries to find a solution that can be expressed simply.

The CSC is listed in figure 6. The algorithm's application on the example problem is illustrated in figure 7.

1. Use greedy set covering to find a small, covering set of partial hypotheses. Attach an empty assignment set, {}, to each partial hypothesis.
2. Give each example an ambiguity score equal to the number of partial hypotheses that cover it. (Unambiguous examples will have an ambiguity score of 1.)
3. Of all the partial hypotheses that cover an example with the best (lowest) ambiguity score, pick one that covers the most examples. Call this *BEST_PARTIAL_HYPO*.
4. Every example is an input/output pair, (x,y). Create a list of all examples that 1) are covered by *BEST_PARTIAL_HYPO* and 2) have the best ambiguity score. Let *POS* be the x components of this set. Let *NEG* be a list of the x components of all examples not covered by *BEST_PARTIAL_HYPO*.
5. Use a concept learner to create a preliminary decision rule that distinguishes the examples in *POS* from those in *NEG*.
6. Add the x component of examples covered by this preliminary decision rule to *BEST_PARTIAL_HYPO*'s assignment set. Remove these examples from consideration. Note that *BEST_PARTIAL_HYPO* is not removed from consideration because a partial hypothesis can become *BEST_PARTIAL_HYPO* more than once.
7. Until no examples remain, repeat at step 2.
8. Create an assignment list by ordering the partial hypotheses according to the size of their assignment sets (that is, a partial hypothesis to which two examples has been assigned is listed before a partial hypothesis to which one example has been assigned). Return this assignment list.

Figure 6: The Conceptual-Set-Covering Algorithm

1. Greedy set covering finds three partial hypotheses, $f(x) = -1$, $f(x) = x$, and $f(x) = \sin\left(\frac{\pi}{2}x\right)$
2. The ambiguity score of each example is computed. Examples $(-5, 1)$, $(-2, 2)$, and $(2, 0)$ are covered by one partial hypothesis each, and so, have ambiguity score 1.
3. CSC chooses $f(x) = 1$ as the best partial hypothesis because it covers an example, $(-5, 1)$, with ambiguity score 1 and because it covers a total of three examples.
4. *POS* is $\{-5\}$. *NEG* is $\{-2, 2\}$.
5. PLS-LISP, a concept learner, is given $POS = \{-5\}$ and $NEG = \{-2, 2\}$. It returns preliminary decision rule $x < -3$.
6. Example $(-5, 1)$ is added to the assignment set of partial hypothesis $f(x) = 1$. Because the assignment set was empty, it is now $\{-5\}$. Example $(-5, 1)$ is removed from consideration.
7. The second time around: *BEST_PARTIAL_HYPO* is $f(x) = -x$. *POS* is $\{-2\}$. *NEG* is $\{2, 5\}$. PLS-LISP returns $x < 0$. So, $(-2, 2)$ and $(-1, 1)$ are assigned to $f(x) = -x$ and removed.
   The third time around: *BEST_PARTIAL_HYPO* is $f(x) = \sin\left(\frac{\pi}{2}x\right)$. *POS* is $\{2\}$. *NEG* is $\{\}$. PLS returns *TRUE*. Examples $(2, 0)$ and $(5, 1)$ are assigned to $f(x) = \sin\left(\frac{\pi}{2}x\right)$.
8. The result is the assignment list shown in table 1b. The assignment-list-to-decision-list algorithm (figure 3) is run on this assignment list. The output is:

```
if x<1 and x≥-3 then -x
else if x≥-1 then sin(π/2 x)
else 1
```

Figure 7: The Conceptual-Set-Covering Algorithm Applied to the Example Problem of Figure 1.

4 Evaluation

The hypotheses produced by CSC are as simple or simpler than those produced by greedy set covering. Two series of tests were used to see if this increased simplicity (as measured by the number of disjuncts is the final hypothesis) would translate into more effective learning (as measured by accuracy on validation test sets). The first test series was designed to test whether CSC could perform significantly better than greedy set covering. Every test in the first test series started with a randomly generated target problem. Targets were generated according to the methods and distributions described in section 2.3. A validation test set containing 100 examples was generated. Multiple training sets of size 5, 10, 20, 50, 100, 150, and 250 were generated. For each size, 29 training sets were generated. Four learners were tested. They are listed in table 2. The 95% confidence intervals were determined using the t-distribution. Figure 8 shows the results of one such test. With training sets of size 100 and larger, CSC performed significantly better than the other learners.

Table 2: The Four Learners Tested — Each consists of an assignment algorithm and a concept learner.

Name	Assigner	Concept Learner
CSC	CSC	Decision-tree learner with no pruning [Quinlan, 1986; Rendell, 1986]
Greedy0	greedy set covering	Decision-tree learner with no pruning
Greedy3	greedy set covering	Decision-tree learner with pruning parameter t_α set to 3.0 [Rendell, 1986]
Most Freq	all examples to the most covering partial hypothesis	Decision-tree learner with no pruning

The first test series showed that CSC could perform better than greedy set covering. The second test series was designed to find where each algorithm was best. The space of possible targets was surveyed. The number of true partial hypotheses was set to 2, 5, or 10. The expected number of additional coincidental partial hypotheses was thus 3.5, 17.0, 59.52, respectively. The number of disjuncts per true partial hypothesis was set to 1, 3, or 5. And the coincidence probability was set to 0%, 5%, 10%, 20% or 50%. At each point in target space, three targets were generated. For each target, training sets of size 5, 10, 20, 50, 100, 150, and 250 were generated. Thus, altogether 945 ($3 \times 3 \times 5 \times 3 \times 7$) tests were conducted.[4]

Because the complexity of the targets varied widely, the average learning curve had high variance. Instead of computing the average, a better statistic can be computed from the rank ordering of the four learners on each learning problem (that is, which learner did best, which did second best, which did third best, and which did worst). The

4 In fact, each test was repeated with 5 different tests sets. The results from these 5 subtests were averaged to produce a single result for each of the 945 tests.

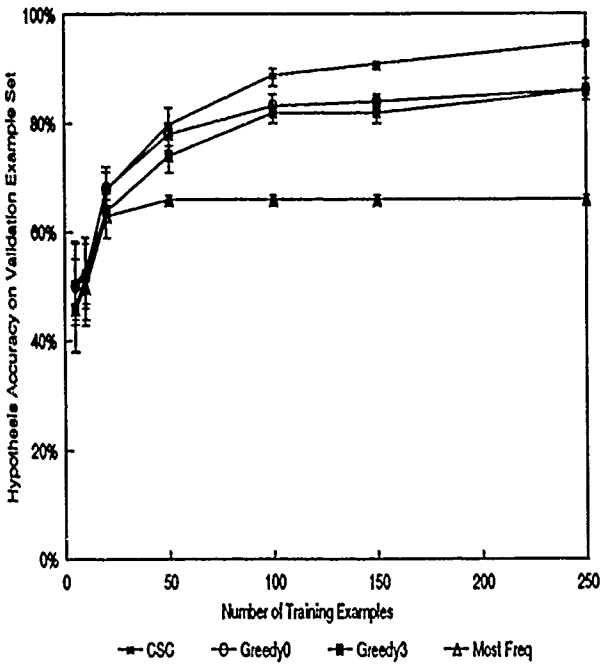


Figure 8: Learning Curves for Four Assignment Algorithms — The target was randomly generated with 5 true partial hypotheses, 3 disjuncts per true partial hypothesis, and a 20% coincidence probability.

statistical significance of the ranking can be evaluated with the Friedman F_r-test for randomized block design with α set to 5% [McClave and Dietrich, 1985].[5]

Figure 9 shows that CSC did best, followed by Greedy0, Greedy3, and Most Freq. Under the assumption that the tests are representative, the Friedman F_r-test indicates that the performance difference between CSC and Greedy0 is significant to at least the 99.5% level.

The performance of the learners was also analyzed at each surveyed point in target space. As section 2.3 detailed, target space is described with three parameters the number of true partial hypotheses in the target, the number of disjuncts per true partial hypothesis, and the coincidence probability. At no surveyed point in target space did another learner do significantly better than CSC. At many points CSC was ranked first and did significantly better than the second ranked learner (see table 3).

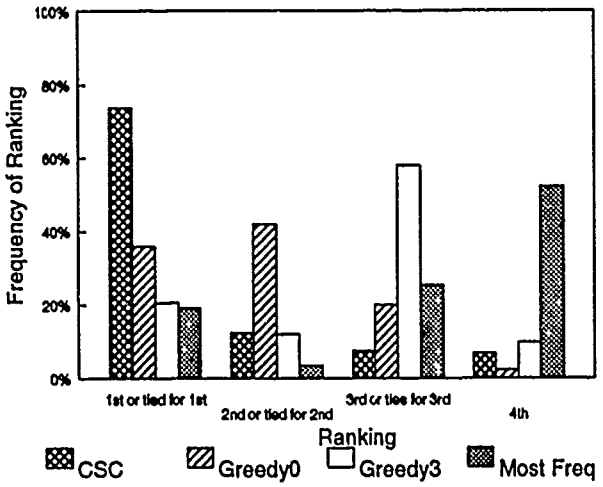


Figure 9: Frequency of Each Ranking for Each Learner — CSC was ranked first or tied for first on 74% of the tests. The second best learner, Greedy0, was ranked first or tied for first on about 36% of the tests.

Table 3: Observed Points in Target Space Where CSC Did Significantly Better than the Number Two Rated Learner — At no point in target space did another learner perform significantly better than CSC.

# of true partial hypotheses	# of disjuncts per true partial hypothesis: 1	3	5
2	-	10-20%	5%,20-50%
5	5-50%	5-20%	5%,50%
10	10-20%	5%	5-20%

5 Discussion and Conclusion

This paper identifies the assignment problem as an important but overlooked part of fit-and-split learning. It contributes both a definition and a model. The characterization explains why poor assignments result in a poor final hypothesis.

5 This statistical test is often used to decide if a new medical treatment is significantly better than an established treatment. It is nonparametric, that is, it requires no assumption about distribution.

Concept Set Covering (CSC) is a new algorithm that tries to make intelligent fit-and-split assignments. It works first with the most unambiguous examples (that is, those examples covered by the fewest partial hypotheses). The resulting preliminary decision rules often remove ambiguous examples from consideration.

CSC was evaluated on hundreds of learning problems. The complexity of the learning problems was varied systematically along three axes: 1) the number of true partial hypotheses, 2) the number of disjuncts per true partial hypothesis, and 3) the coincidence probability. At many points in problem space, CSC did significantly better than alternative methods. It never did significantly worse. Thus, the empirical evaluation suggests that (at least to the extent that the hundreds of learning problems were representative) CSC is the better algorithm.

Several limitations of the fit-and-split method and the CSC algorithm should be kept in mind. First, although the fit-and-split method is a simple way to decompose a learning problem, it may not always be the best. Sometimes it might be better to first split and then fit [Tcheng, et al., 1989]. Also, because the choice of partial hypotheses (fitting) can affect the quality of the decision rules (splitting), it might sometimes be best to integrate the fitting and splitting processes. The CSC algorithm also has limitations. It is heuristic; its performance is not guaranteed. Indeed, the empirical evidence indicating that it performs well assumes that the model produces representative assignment problems. The model was chosen for its simplicity and because it seems to fit, at least as a first approximation, problems such as scientific discovery. Study of other fit-and-split problems might lead to different models and different heuristic algorithms. Also, CSC works only on all-or-nothing partial hypotheses. It does not work with probabilistic partial hypotheses ("the probability that this partial hypothesis covers this example is 0.9"). Probabilistic partial hypotheses is an area of future research.

Despite these limitations, the fit-and-split method and CSC offer important benefits. The fit-and-split method is a straightforward and efficient way to simplify learning problems. CSC, according to the empirical evaluation, works significantly better than assignment methods currently in use. In addition, CSC is widely applicable. Thus, the use of the CSC assignment algorithm is likely to improve any fit-and-split learning system.

References

[Cohen and Feigenbaum, 1982] P. Cohen and E. Feigenbaum. *The Handbook of Artificial Intelligence.* Volume 3, HeurisTech Press, Stanford, CA, 1982.

[Drastal et al., 1989] George Drastal, Stan Raatz, and Gabe Czako. Induction in an abstract space. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, 1989.

[Falkenhainer and Michalski, 1986] Brian Falkenhainer and Ryszard S. Michalski. Integrating qualitative and quantitative discovery: the ABACUS system. *Machine Learning*, 1(4):367-401, 1986.

[Garey and Johnson, 1979] M. R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, San Francisco, 1979.

[Kadie, 1989] Carl M. Kadie. *Diffy-S: Learning Robot Operators from Examples of Operator Effects.* Technical Report UIUCDCS-R-89-1550, Computer Science Department, University of Illinois, Urbana, IL, October 1989. Masters Thesis.

[McClave and Dietrich, 1985] James T. McClave and Frank H. Dietrich. *Statistics.* Dellen Publishing Company, San Francisco, third edition, 1985.

[Michalski, 1983] R. S. Michalski. A theory and methodology of inductive inference. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, chapter 4, pages 83-134, Palo Alto: Tioga Press, 1983.

[Michalski and Chilausky, 1980] R. S. Michalski and R. L. Chilausky. Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing and expert system for soybean disease diagnosis. *Policy Analysis and Information Systems*, 4(2), 1980.

[Pitt and Reinke, 1988] L. Pitt and R. E. Reinke. Criteria for polynomial time (conceptual) clustering. *Machine Learning*, 2(4):371-396, 1988.

[Quinlan, 1986] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1), 1986.

[Rendell, 1986] Larry Rendell. Induction, of and by probability. In L. Kanal and J Lemmar, editors, *Uncertainty in Artificial Intelligence*, pages 429-443, New York: North Holland, 1986.

[Shen and Simon, 1989] Wei-Min Shen and Herbert A. Simon. Rule creation and rule learning through environmental exploration. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 675-680, Detroit, MI, 1989.

[Stepp and Michalski, 1986] R. E. Stepp and R. S Michalski. Conceptual clustering: inventing goal-oriented classifications of structured objects. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, Volume II*, chapter 20, pages 471-498, Los Altos: Morgan Kaufmann, 1986.

[Summers, 1977] P. Summers. A methodology for LISP program construction from examples. *J. ACM*, 24, January 1977.

[Tcheng et al., 1989] David Tcheng, Bruce Lambert, Stephen Lu, and Larry Rendell. Building robust learning systems by combining induction and optimization. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 806-812, Detroit, MI, 1989.

Incremental Learning of Rules and Meta−rules

Marc Schoenauer

Equipe Numérique Symbolique,
Centre de Mathématiques Appliquées
Ecole Polytechnique
91128 Palaiseau France.

Michèle Sebag

Laboratoire de Mécanique des Solides
Ecole Polytechnique 91128 Palaiseau.
and, Equipe I&A, LRI Bat 490, Paris-XI Orsay
91134 Orsay France.

Abstract

This paper presents a general incremental learning scheme : a single generalization algorithm can both learn a set of rules from a set of examples, and achieve the refinement of a previous set of rules. This approach is based on a redescription operator called reduction : from a set of examples and a set of rules, we derive a new set of examples describing the behavior of the rule set. New rules are extracted from these behavioral examples : those rules can be seen as meta-rules, as they control previous rules in order to improve their predictive accuracy.

1. Introduction.

This paper deals with incremental learning from examples. Two kinds of incremental processes are distinguished :

First, an algorithm that sequentially handles the examples is incremental {Lebowitz 87}. This sequential incrementality is useful in case of huge amount of data, in order to avoid exponential explosion.

A second kind of incrementality aims at gradually refining the computed knowledge : for instance, learning by discovery gradually learns numerical laws from examples {Langley & al. 1984, 1986 ; Falkenhaimer Michalski 1986}. In neural networks, the knowledge encoded within the network coefficients is also gradually refined and optimized {Rumelhart & al. 1986 ; Fogelman & al. 1986}. This iterative incrementality is useful when the search space is huge (such as the space of polynoms of several variables) and/or when the problem is untractable by direct ways.

Our aim is to study a general learning scheme, leading to both sequential and iterative incrementality. A 2-steps process is presented :

- In a first step, a set of rules is learnt from a set of examples by a generalization algorithm. The only assumption we make is that this algorithm is able to produce deliberatly under-optimal solutions (rules are redundant, admit exceptions,...). So, the learning problem is given an approximate solution, and the next step is to refine this approximate solution.
- The second step performs the transition toward another learning problem, namely the refinement of the previous set of rules. This transition is performed by a redescription opeator called reduction : given a set of rules and a set of examples describing the learning domain, we derive a set of **examples describing the behavior of the rule set** over the learning domain.

In this new context, generalization applies again : a new set of rules is learnt in order to correct previous rules. Successive applications of this 2-steps process (generalization, reduction) allow more accurate and more complex (because disjunctive) rules to be discovered ; moreover, this process can sequentially handle subsets of examples. So, both sequential and iterative incrementality can be achieved by this learning process, called multi-layers learning.

We outline that reduction applies whatever the generalization algorithm : it transforms the refinement of a learning output into another (boolean) learning problem.

This paper is organized as follows :

In section 2, some incremental learning processes are briefly reviewed. We discuss some problems raised by incrementality, such as the convergence of these iterative techniques, and the stability of the learning output.

Section 3 introduces the notion of symbolic approximation ; the definition of an approximate learning output refers to the defects of a knowledge base, as stated in {Rajamoney DeJong 1987}. The reduction operator is then given : it enables to describe the behavior of a rule set over a learning domain.

Successive applications of generalisation-reduction can be done. But the worthiness of this iterative process depends on the ability of the generalisation to produce under-optimal solutions, in terms of conciseness and exactness. The end of section 3 presents a generalisation algorithm that fullfills these requirements, and that we used to experiment our approach.

Finally, two applications of the reduction are detailed in section 4 : reduction allows to gradually learn a complex knowledge from the whole set of examples. It also allows to sequentially learn from subsets of a large learning set.

2 Examples Of Incremental Processes

This section does not attempt to give an exhaustive state of the art (see for instance {Kodratoff 1989}). Our purpose is to point out on some well-known learning processes, which problems are solved and which are raised by incrementality.

Sequential incrementality mainly addresses the problem of exponential explosion : examples are handled one by one, as a direct global learning is impossible for material reasons.

Iterative incrementality applies when no direct learning algorithm is known : a good solution is gradually built without any new external information.

2.1 Sequential Incrementality.

At each step of a sequential learning process, current knowledge evolves by considering one new example. We focus on the generality of this current knowledge, and more precisely, we consider the evolution of this generality during the learning process.

2.1.1 Monotonous Learning.

If this generality is monotonous (increasing or decreasing), the process is said to be monotonous. An instance of monotonous incrementality is given by the Version Space {Mitchell 1982}. We briefly recall the main features of this famous approach. The space of the most specific versions of the concept to learn, (respectively of the most general versions) denoted by S (resp. G), is initialized to "nothing" (resp. "everything"). S is then continuously generalized to cover new positive examples of this concept while G is continuously specialized in order to reject new negative examples. G and S are ensured to be equal as soon as enough exact positive and negative examples have been encountered[1].

Note that this monotonous approach does not allow any revision of the knowledge : as this process involves no backtrack, no error is allowed. The process is **ensured to converge but the quality of the result is guaranteed only if the data are error-free.**

Another example of incremental monotonous process is the incremental acquisition of concepts by analogy {Vrain Lu 1988}. Similarly, the generality of the current knowledge continuously increases, and same remarks about convergence and error apply.

[1]The Version Space is adapted to the characterization of conjunctive concepts ; moreover it can be adapted to disjunctive concepts as shown in {Manago 1988}.

2.1.2 Non Monotonous Learning.

An instance of non monotonous learning process is the incremental conceptual classification {Decaestecker 1989}. This approach gradually builds a tree of concepts ; the examples are considered one by one. Four operators are used : creation/deletion of a node, splitting of a node into several nodes, fusion of several nodes into one. It is clear that these operators are inversible : hence the learning process may undo what was done, delete a node previously created, etc.

This process is theoretically reversible : it shoud better handle noisy domains than monotonous processes. On the other hand, it follows from its reversibility that it is not ensured to converge : oscillations of the knowledge (here the tree of concepts) may appear. This problem is solved via statistical criterions. The operators (creation, deletion, etc...) are trigg-ered only when some numerical thresholds are reached ; those thresholds depend on past examples.

The consequence is that, as process goes on and thresholds increase, the examples have less and less effects on the structure of the tree: they induce only peripheral evolution of the knowledge. In other words, the learning output may depend on the order of the learning input. This can be seen as the inertia of the learning process {Cornuejols 1989}.

In short, if non monotonous learning is more adapted to noisy domains, it is not guaranteed to converge. A way to ensure convergence is to gradually decrease the influence of examples, that is increase the inertia of the learning process.

2.2 Iterative Incrementality.

The main concern of this type of incremental learning is to gradually provide more accurate knowledge. A learning phase includes two steps : first, current knowledge is elaborated ; second step achieves the transition to the next learning phase.

2.2.1 Neural Networks.

In neural networks, the knowledge is encoded by numerical coefficients {Rumelhart & al. 1986, Fogelman & al. 1986}. The coefficients are gradually optimized ; examples are taken into account in a cyclic way[2]. This process fits into the above scheme : the first step computes new values for the coefficients from current values and current example. The transition step just sets the coefficients to the computed values.

In this case, incrementality is a purely mathematical method, which ensures the almost sure convergence of the process.

2. For this reason, neural networks have been classified as iterative systems : after a while they handle no new information, strictly speaking.

2.2.2 Learning By Discovery.

Learning by discovery gradually discovers numerical laws from examples {Langley & al. 1984, 1986 ; Falkenhainer & Michalski 1986}. For instance, one discovers the law $PV = nRT$ from examples of perfect gaz described according to their pressure P, their volume V, their temperature T and the number of moles n.

Data and knowledge are represented using two levels : the first level is a set of descriptors (P, V, n, T) ; the second level is a set of examples, which are points belonging to this description space (here in 4 dimensions).

In the first step, some descriptors are built (PV, in the example above) because of their steadiness on some subset of examples ; those new descriptors are functions of the previous descriptors, and hence are computable on the set of examples.

In the transition step, the set of descriptors is updated, and so is the description of the examples. We now have examples belonging to the 5 dimensions space, described by (P, V, n, T, PV), and a next learning phase is possible, leading to PV/T, and finally to PV/nT.

The process is data-driven : the descriptor PV/nT induces a law, as it takes the constant value 8.32 = R on the data (if only perfect gaz are described). In this case, incrementality allows to heuristically explore a huge search space, the polynomial fractions of several variables and of any degree.

2.3 Some Remarks.

On sequential learning : we notice that the processes we have reviewed only handle one example at the time. For the sake of robustness, a (small) subset of examples could be considered in each learning step : the effects of noise would be diluted.

On iterati.. learning : finding a solution and refining this previous solution are achieved by a single algorithm. This approach relies on the transition step which transforms the initial problem by use of current knowledge. This method is well-known in the field of arithmetic calculus ; so it is not surprising, if it is applied to learn numerical knowledge. In the symbolic field, the question is : what could be the semantics of such a transition ? How examples could be transformed using current knowledge, in order to refine this current knowledge ?

This paper attempts to answer these remarks: a transition mechanism, inspired from the learning by discovery, is adapted to the learning from examples {Michalski 1984 ; Kodratoff 1988}.

3 Learning And Approximation.

Our aim is to use the same generalization algorithm, both to find a solution at one learning phase, and to refine this solution at the next learning phase.

A solution is to be refined, only if it is an approximate solution : we first discuss what we mean by approximation (§3.1). The transition we propose from one learning phase to the another is detailed in §3.2. Conditions of application are studied in §3.3. We outline that the presented approach could work with any generalization algorithm satisfying hypotheses §3.3. But we briefly recall the generalization we used to illustrate the value of our approach (more details may be found in {Sebag Schoenauer 1988, 1989}).

3.1 Symbolic Approximation.

A solution to a learning-from-examples problem is provided by a set of discriminant rules {Quinlan 1986 ; Michalski 1986}. We suppose this solution needs to be refined, and call it an *approximate solution*. This notion refers to the defects of a knowledge base, as stated by {Rajamoney & DeJong 1987}.

For instance, let a *very* approximate knowledge base be given by the four rules :

r_1: *If I walk on a snake, danger*
r_2 : *If I see 3 black crows, danger*
r_3 : *If I walk on a grass snake, no danger*
r_4 : *If I have my mascots, no danger.*

This knowledge base has all kinds of defects:

- It is <u>incomplete</u>, as no conclusion is delivered (no rule is fired) if *I see only 2 black crows*.
- It is <u>inconsistent</u>, as contradictory conclusions are delivered if *I walk on a cobra and I have my mascots* (r_1 concludes to *danger* and r_4 concludes to *no danger*).
- It includes <u>errors</u> : no danger if *I walk on a dead snake,* in spite of rule r_1.
- It is <u>redundant</u> : *I may both walk on a snake and see 3 black crows.*

An approximate rule set is just a set of incomplete, inconsistent, erroneous and/or redundant rules. Notice these defects are not independent : when the number of rules increases, then the incompleteness of this rule set is likely to decrease. But unfortunately inconsistency and errors are likely to increase in this case : the chance for an example to fire at least one rule increases, but the chance to fire two contradictory rules increases too.

Many recent works attempt to find a balance between incompleteness and inconsistency ; see {Quinlan 1986, Michalski &al. 1986, Clark Niblett 1987, Ganascia 1987} for generalization of rules with exceptions ; see {Quinqueton 1983; Cestnik & al. 1987 ; Gams 1988} ; see also §3.4.

From the incrementality point of view, inconsistency seems to be a more tractable defect than incompleteness. As a matter of fact, when an example fires no rule, no supplementary information about this example is provided by previous learning results. On the opposite when an example fires contradictory rules, some effective work is made, though not yet sufficient. New information is given about this

example (the list of fired rules), and a next learning problem can be set : learning to solve the previous rules inconsistencies.

3.2 Resolution Of Inconsistencies.

3.2.1 Numerical Resolution.

A simple way to solve inconsistencies is numerical : the final conclusion is given by a majority vote, where every fired rule votes for its conclusion. This method can be refined by weighting the rules. Statistical criterions can be used in order to optimize the reliability of this decision process {Lerman 1989, Perron 1989}.

Obviously, weights give a partial order on the rule set. However, this numerical representation of priorities is not adapted : priority is not necessarily a transitive property (a rule r1 may be preponderant over a rule r2, which may be preponderant over a rule r3, which may in turn be preponderant over rule r1...). More generally, priorities of rules heavily depend on the context : a symbolic treatment should better suit our purpose.

3.2.2 Symbolic Resolution.

Let us see how this can be done, using generalization again.

Let us go back to our approximate rule set about snakes and danger, and consider an example described by :

I walk on a cobra, and it is full moon, and I have my mascots (if the conclusion is known, it is likely *danger*).

The above description may be transformed according to the rule set. It gives :

- r_1 *applies (I walk on a snake)*
- r_3 *does not apply (I do not walk on a grass snake)*
- r_4 *applies (I have my mascots).*

So, from an example of the learning domain, (with known description and conclusion) an example of the behavior of the rule set is derived :

(description) r_1 *applies,* r_3 *does not,* r_4 *does,*

(conclusion) *danger.*

From such new examples, new rules can (and will) be learnt, for instance :

If r_1 *applies and not* r_3, *then danger* (in other words, *if I walk on a snake which is not a grass snake, danger*).

Remark : this redescription mechanism provides an alternative to asking the experts to solve inconsistencies. As the original conclusion of the example remains unchanged by the redescription, we now have **examples of conflicts arising among rules, together with the actual conclusion.** So, resolution of inconsistencies is learnt from reduced examples by the system, rather than asked to the experts.

3.2.3 Definition of Reduction.

Let us more formally define the redescription above. Let Ω denote the current description space of the learning domain, and B be a set of rules expressed within Ω. We wish not to depend on the actual formalisation of Ω (boolean logic, multi-valued propositionnal logic, predicate logic, symbolic objects,...).

We only assume that for any rule expressed within Ω, for any example expressed within Ω, we are able to know if this example fires this rule, i.e. if the description of the example satisfies the premises of the rule (this requirement is fulfilled if the rule is to be of any use). In other words, a rule defines a boolean descriptor on Ω : for any example, the premises of the rule are or aren't satisfied.

Definition 1.
The reduction is defined according to a rule set B, *and denoted* π_B. *The reduction is a redescription operator defined from* Ω *into the boolean space of dimension L, if L is the number of rules in B.*

$$\pi_B : \Omega \longrightarrow \{0,1\}^L$$

$$\pi_B : s \in \Omega \longrightarrow \pi_B(s) = \{r_j(s), j = 1..L\}$$

where the reduced descriptor r_j is given by :

$$r_j(s) = \begin{cases} 1 \text{ if } s \text{ satisfies the premises of the } j\text{-th rule in } B \\ 0 \text{ otherwise.} \end{cases}$$

This redescription allows to transform any example set A, expressed within Ω. An example in A is given by its description s belonging to Ω, and its conclusion *Conc(s).*

Definition 2
From the learning set A *according to the rule set* B, *the learning reduced set denoted* A_B *is given by* :

$$A_B = \{ (\pi_B(s_i), Conc(s_i)) \,/\, (s_i, Conc(s_i)) \in A \}$$

The reduced learning set A_B describes the behavior of B on the examples in A ; it is expressed in boolean logic, whatever the initial representation of A and B. The next move is obviously to generalize the reduced learning set A_B ; but it requires the reduced set to be a valuable learning set.

3.3 Conditions Of Application.

A learning set is valuable if it enables to learn a good set of rules : a learning set is found to be valuable a posteriori. A priori, one could just say that a valuable learning set ought to be sufficiently consistent, and to include enough information ; those characteristics are vague indeed.

So we have to suppose that the initial learning set A does fulfill those good properties. The question now is : does the reduced learning set still have those properties, or, in other words,

how does the reduction affect the consistency and the quantity of information of the data ?

Notice that the reduction is not necessarily injective : distinct examples can have the same redescription through π_B. So reduction can create inconsistencies, if examples of distinct conclusions are given the same image. Consistency is easy to check : two examples of distinct conclusions must be discriminated by at least one rule in B, i.e. there must exist a rule which covers exactly one of both examples.

The number of examples in the reduced learning set A_B is less than the number of examples in the initial learning set A. But the reduced learning set must still contain enough information, in order to enable a further learning phase : the number of examples should not decrease too much. (This rough criterion about sufficiency should be refined with respect to the size of the description space).

One shows that both conditions are fulfilled if the rule set is sufficiently redundant : as the number of rules increases, the reduction beomes injective ; it follows that the number of examples remains and that reduction causes no inconsistency.

In the following, we suppose that the reducion π_B preserves the consistency and the amount of information of the learning set : this is ensured by tuning the redundancy of the rule set B, as allowed by several heuristics of generalization {Cestnik & al. 1987, Gams 1988}, and by the generalization below.

3.4 Generalization And Heuristics.

Our approach of incrementality does not depend on the generalization algorithm provided that sufficiently redundant rules can be learnt. Nevertheless, we briefly recall here the generalization we used, as the whole process generalisation-reduction actually works with it, as will be presented in the last section. A detailed presentation is given in {Sebag Schoenauer 1988, 1989}.

3.4.1 Generalization By Elimination.

The following algorithm is a star-like generalization {Michalski 1984}, which handles multi-valued propositionnal logic. Its main originality is a logical pruning of counter-examples, based on the near-miss notion {Winston 1975, Kodratoff & Loisel 1984}.

Let us take an example ; a toy learning set about right and wrong quests is given by :

Ex 1 : My name is Arthur, and my favorite color is red, and I am good ; quest is right.

Ex 2 : My name is Triboulet, and my favorite color is blue, and I am good ; quest is wrong.

Ex 3 : My name is Ganelon, and my favorite color is yellow, and I am bad ; quest is wrong.

The star-like generalization handles the examples one by one ; from one example, one attempts to find maximally discriminant rules, rejecting counter-examples.

Let us consider example 1 ; we restrict to the dropping rule generalization for the sake of simplicity. We want to know which conditions can be dropped or, as well, which conditions cannot be dropped. From example 2, it follows that conditions *My name is Arthur* and *my favorite color is red* cannot be dropped simultaneously : the rule *If I am good then quest is right* does not reject example 2.

Given example 2, example 3 gives no information : if conditions *My name is Arthur* or *my favorite color is red* are kept, it ensures the reject of example 3, whose name is not *Arthur* and whose favorite color is not red. In short, if example 2 is discriminated by a rule generalizing example 1, then example 3 is discriminated too : example 3 is useless for the discriminant generalization purpose, and it can be pruned.

More formally, let s be an example of an example base A. Any counter-example t of A gives a constraint over the generalization of s the descriptors which discriminate t from s c not be dropped simultaneously. The constraint $C(s,t)$ is a subset of integers, given by

$C(s,t) = \{i$ / attribute i discriminates s and $t\}$

We say a counter-example t_0 is a maximal near-miss to s in A if the constraint $C(s,t_0)$ is minimal for the set inclusion, among all $C(s,t)$, t counter-example of A.

We then search all minimal subsets of integers M, such that M intersects every constraint $C(s,t)$. From such a set M, a rule $r_{s,M}$ is defined as follows : its premises are the conjunction of all conditions in s regarding attributes in M ; its conclusion is the conclusion of s. We prove that $r_{s,M}$ is a maximally discriminant generalization of s : by construction, for any counter-example t discriminated from s, there exists an element in $C(s,t)$ which belongs to M : the corresponding attribute allows to discriminate s and t ; this condition is kept from s to $r_{s,M}$, hence $r_{s,M}$ still discriminates t.

The search of subsets M is achieved by a graph exploration, exponential with respect to the number of constraints. However, it is enough for a subset M to intersect all constraints $C(s,t)$, for t maximally near-miss to s. This generalization by elimination so reduces the size of exponential exploration by a preliminary (polynomial) pruning.

3.4.2 Heuristics.

Two problems are raised by the above algorithm.

First, it finds out perfectly discriminant rules ; this is useless and undesirable if the example set is noisy, as stated in {Clark Niblett 1987}.

Second, many maximally discriminant rules are found and some selection is required.

Four heuristic parameters are introduced to overcome these problems. Those heuristics are

parameterized by the expert, according to the features of available data :

- The exception rate ε is related to the noise of the data. A uniform forgetting rate of ε is applied, when constraints are built and explored to find discriminant rules. By this way, rules which admit exceptions (do not reject counter-examples), may be found. One requires the ratio of exceptions for a rule to be less than ε (criterion 1). ε belongs to [0, 1].
- The significance threshold α is related to the redundancy of the data. One requires a rule to generalize more than α examples (criterion 2). α is an integer.
- The redundancy rate τ is related to the sufficiency of the data. The idea is that, if data are sufficient, only the "best" rules should be kept ; one then requires the number of examples covered by a rule r to be the maximal number of examples, covered by a rule generalizing the current example s. But this criterion leads to instability when data are insufficient. So we require the number of covered examples to be greater than the product of τ by this maximal number of examples covered by a rule generalizing s (criterion 3). τ belongs to [0,1].
- Finally, a more technical parameter, the patience rate F (like Forget), enables to control the computational effort. When a given number of failures consecutively occur during the generalization of an example (a rule fails because of criterion 1, 2 or 3), then the exploration is speeded up by randomly forgetting of constraints.

Criteria 1 and 2 are largely used in the litterature; criterion 3 is also well studied. Criterion 4 is inspired from simulated annealing {Davis Steenstrup 1987} (the patience of the learner stands for the temperature of the cristal); as far as we know, this is a new feature to generalization area.

4. Multi–Layers Learning.

In this section, we define a general incremental learning scheme which uses two operators, generalization and reduction (§4.1). This scheme allows to gradually learn more accurate and complex (disjunctive) rules (§4.2). It also allows to sequentially handle the examples (§4.3). Some comparative results are presented on a well-studied learning set {Michalski & al. 1986, Clark Niblett 1987, Cestnik & al. 1987}.

4.1 A Learning Layer.

The reduction gives from a learning set A and a rule set B, a new learning set A_B. On the other hand, many generalization algorithms provides a rule set B from a learning set. Reduction and generalization may then be seen as operators, and combined ; the 2-steps process (generalization of a rule set, reduction according to this rule set) is called a learning layer.

4.1.1 Refinement Of Previous Rules.

We claim that the generalization on reduced learning set A_B implicitly achieves the refinement of rule set B :

- First, if a rule in B has a good predictive accuracy, this information is implicitly available from the reduced learning set A_B. As a matter of fact, if a rule has a good predictive accuracy, it is often fired : the reduced descriptor often takes the value 1, and in this case, the actual conclusion is often equal to the rule conclusion. Hence, there is a correlation in A_B between a value of this descriptor and a value of the conclusion. So the rule will be discovered again by next generalization. This process is stable, as good rules in B are kept.

- Second, the same argument ensures that irrelevant rule are dropped : if a rule is irrelevant, the associated reduced descriptor is irrelevant with respect to A_B too. As generalization is supposed to detect and forget irrelevant descriptors, this guarantees that the rules learnt from A_B do not mention previous irrelevant rules.

- Third, generalization discovers links among descriptors and conclusion. In the reduced learning set A_B, examples are described according to the rules in B they trigger. Hence, the triggering of rules can be generalized from A_B : the generalization solves conflicts arising among previous rules.

4.1.2 Approximation Requirements.

In multi-layers learning, rules have a double part : they are provided by a learning phase, as a solution to the current learning problem ; then they derive descriptors, and at next phase those descriptors are inputs to the next learning problem. Desirable properties about learning solution and means are rather different : one wants to learn concise and general rules ; but to this aim, one needs sufficient, precise, various descriptors... From an incremental point of view, we obviously prefer to enable the further learning phase, than to optimize the current learning output : hence in a learning step, the generalization must be able to provide redundant and inconsistent rules.

This is a general strategy of approximation : as shown in genetic algorithms or simulated annealing {Holland & al. 1985}, non-optimal solutions must be kept especially in first steps, in order to avoid local optima.

4.2 Iterative Incrementality.

We suppose here the data can be handled by the generalization in a reasonnable time (which depends on the available machine, the available generalization and the patience of the experts...).

4.2.1 Outline.

An iterative learning is performed by multi-layers learning as follows :

Learning phase 1 :

1.1- Generalize a rule set B_1 from the initial learning set A.

1.2- Reduce A according to B_1 : $A_1 = A_{B1}$.

Learning phase 2 :

2.1- Generalize[3] a rule set B_2 from reduced learning set A_1.

2.2- Reduce A_1 according to B_2 :

$$A_2 = (A_1)_{B2} = A_{B1.B2}$$

Learning phase 3:

3.1- Generalize B_3 from A_2, and so on.

The only condition for this scheme to apply is for the rule set B_i to be sufficiently redundant so that the reduction derives a sufficient and consistent learning set A_i from A_{i-1} : as mentionned in §3.3, the generalization must be able to provide sufficiently redundant rules.

This process leads to a sequence of rule sets, called multi-layers rules : premises of rules in B_{i+1} are conjunction of litterals, those litterals being the premises of rules in B_i, **or negation of theses premises.** Rules in B_{i+1} may be seen as meta-rules with respect to B_i, as they perform some control over B_i :

r_l : *If r_i and r_j are fired, and r_k is not, then,...*

where r_l *belongs to* B_{i+1},

r_i, r_j *and* r_k *belong to* B_i.

Obviously rules in B_i can always be expressed with respect to the original descriptors used in B_1 ; however, rules in B_i, $i > 1$, may be disjunctive with respect to the original descriptors, because negation of previous conjunctive premises are used.

We point out that there is no generality relation among rules in B_i and rules in B_{i+1}, hence this learning process is not monotonous (see §2.2.1) : premises of rules in B_{i+1} are not necessarily more general nor more specific than premises in B_i.

Evolution of this process is both expert- and data-driven. The expert adjusts the parameters at each generalization step, especially the redundancy rate (this adjustment is discussed in the following). The process stops when the current learning layer gives no improvement, with respect to the predictive accuracy on a test learning set. In the experimentations we made, predictive accuracy can significantly increase from 1 to 2-layers rules, can still increase from 2 to 3-layers rules, and then usually decreases as the number of layers increases.

4.2.2 Validation.

Multi-layers learning (*ML2*) has been implemented in Language C on a Unix work-station. It uses the generalization algorithm described in §3.4. The results obtained on a well-studied medical learning set (about prognosis of breast cancer) are given below. The reference results are predictive results obtained by algorithms *AQR* {Michalski & al. 1986}, *Assistant 1986* {Cestnik & al. 1987} and *CN2* {Clark Niblett 1987} ; those results are found in the latter paper, as well as the results of a simple bayesian classifier, denoted *Bayes*.

Table 1 : Reference Results.

Results on : obtained by	Training set (200 ex.)	Test set (86 ex.)
Bayes	*70%*	*65%*
AQR	*100%*	*72%*
CN2	*72-76%*	*70-71%*
Assistant 86	*85-92%*	*62-68%*

The results of *ML2* are estimated the same way. A *n*-layers rule set is said to be optimal if its predictive accuracy on the test set is maximal among *n*-layers rule sets.

Here are the results obtained for optimal 1-layer rule set B, for optimal 2-layers rule sets (C_1, C_2) and for optimal 3-layers rule sets (D_1, D_2, D_3).

Those rule sets are learnt with given parameters of exception rate ε and redundancy τ.

Table 2 : Optimal Results Of Multi-Layers Learning

	Train. set	Test set	ε	τ
B	*66%*	*62%*	*.7*	*.3*
C_2	**93%**	**75%**	*.2*	*.4*
C_1	*79%*	*58%*	*.5*	*.3*
D_3	**94%**	**78%**	*.3*	*.7*
D_2	*83%*	*60%*	*.4*	*.7*
D_1	*62%*	*48%*	*.7*	*.6*

One notices that the best $n+1$-layers rule set are based on a *n*-layers rule set which is not the best one among *n*-layers rule set.

The tuning of generalization parameters is exponential with the number of layers. But some explorations of these possibilities leads to the following (experimental) conclusions : to build an optimal *n*-layers rule set, one must admit a high exception rate and redundancy at the beginning ; generalization must gradually look for more concise and exact rules. As in simulated annealing again, optimality requirements must be gradually raised to finally give a good solution.

3. The same generalization algorithm applies than in first phase : the reduced learning set is expressed in boolean logic. Any generalization algorithm should be able to cope with this minimal representation.

4.3 Sequential Incrementality.

In this section, we suppose the learning set is very large. The problem is then to avoid exponential explosion. Multi-layers learning can perform sequential incrementality as follows :

- Some subsets A_1, ..A_L are extracted from the initial learning set A by the experts. Those subsets may be a partition, or may overlap.

Learning phase 1 :

- Generalization of learning set A_1 gives a rule set B_1.
- All subsets A_i are reduced according to B_1. One sets :

$$A_i^1 = (A_i)_{B1}$$

Learning phase 2 :

- Generalization of reduced learning set A_2^1 gives a rule set B_2.

More generally, one has at phase $j+1$:

$$A_i^j = (A_i)_{B1.B2...Bj}$$

$$B_{j+1} = \text{Generalization } (A_{j+1}^j).$$

This approach of sequential learning allows to consider a set of new examples at each learning phase (see §2.3), and the size of these sets is here controlled by the expert.

The limitation of this process is the following: if two consecutive subsets are too different, then rules learnt from the first subset fails to provide a good description of second subset. This problem is encountered whenever consecutive learning subsets are non-homogeneous ; in this case, some initial descriptors must be added to reduced descriptors, in order to preserve consistency of the reduced description. This is done by experts at the moment.

4.4 Conclusion.

We state a general incremental learning scheme, called multi-layers learning. The general idea is to first learn an approximate rule set, rubbing the possible defects of the learning set itself (noise, incompleteness,...) and then to learn to refine a previous rule set.

This approach relies on the reduction operator, which performs the transition from first to second learning problem. A new description of the learning domain is derived from an approximate rule set. Within this new description, initial examples provides examples of the behavior of the rule set over the learning domain. From these behavioral examples, generalization can extract new rules, which correct defects and inconsistencies of previous rules. A sequence of rule sets is so gradually built.

Only one assumption is made about generalization : it must be able to provide redundant rules. Under this requirement, any generalization can be used together with reduction : reduction defines boolean learning problems, which can always be handled by generalization (boolean logic is a minimal representation).

Multi-layers learning allows to gradually learn more accurate rules. Validation on a well-studied learning set shows a slight predictive improvement, compared to some famous non-incremental learning algorithms.

This process also enables to sequentially handle subsets of examples. Compared to handling the examples one by one, this sequential incrementality is more flexible (the size of those learning subsets is controlled by the expert) and suited to noisy domains.

The limitation of this approach comes from the roughness of the reduction. The transition to boolean logic is a bit abrupt, when the initial data are expressed within predicate logic : many 1^{st} order rules are required to give a sufficient 0^{th} order description. An extension of reduction, from boolean to multi-valued propositionnal logic {Michalski 1975}, is planned.

4.4.1 Acknowledgements.

We thank J. Zarka, L.M.S., Ecole Polytechnique, who made this study possible. Long discussions with Y. Kodratoff L.R.I Orsay, and E. Diday, Paris-IX & I.N.R.I.A., greatly helped to clear and improve this paper. The data about breast cancer have been gathered by the Universisty Medical Center, Ljubljana, Yugoslavia. Thanks to M. Zwitter and M. Soklic ; many thanks also to D. Aha, University of California, Irvine, who kindly broadcasts these data and many more.

4.4.2 References.

B. Cestnik, I. Bratko, I. Kononenko, ASSISTANT 86 : A Knowledge Elicitation tool for sophisticated users. *In Progress in Machine Learning, EWSL 87, I. Bratko & N. Lavrac Eds.*

P. Clark, T. Niblett : Induction in noisy domains. *In Progress in Machine Learning, EWSL 87, I. Bratko & N. Lavrac Eds.*

L. Davis, M. Steenstrup : Genetic Algorithms and Simulated annealing : An Overview, *in Genetic Algorithms and Simulated Annealing,* L. Davis Ed., Pitman London (1987).

C. Decaestecker, Concept formation via an adequacy criterion, *Data Analysis/Learning symbolic and numerical knowledge, E. Diday Ed, Novasciences Antibes, 1989.*

B.C. Falkenhainer & R.S.Michalski : Integrating quantitative and qualitative discovery in the ABACUS system. *In Machine Learning April 1986.*

F. Fogelman, P. Gallinari, Y. Le Cun, S. Thiria : Network learning *in Machine Learning : An Artificial Intelligence Approach, Y Kodratoff, R.S. Michalsky, Eds, Morgan Kauffman 1986.*

M. Gams : A new breed of knowledge acquisition system, using redundant knowledge, *Proceedings EKAW BONN 1988.*

J.G. Ganascia : AGAPE et CHARADE, deux techniques d'apprentissage symbolique appliquées à la construction de bases de connaissance. Thèse d'Etat, mai 1987, Orsay.

J.H. Holland, K.J. Holyoak, R.E. Nisbett, P.R. Thagard : Classifier systems, Q-morphisms, and Induction, *in Genetic Algorithms and Simulated Annealing, L. Davis Ed., Pitman London, 1985.*

Y. Kodratoff, R. Loisel : Learning complex structural descriptions from examples, *in Computer vision, graphics and image processing, Vol 27, 1984.*

Y. Kodratoff : Introduction to Machine Learning, *Pitman Publishing, 1988.*

Y. Kodratoff : Characterising machine learning programms : a european compilation, *Rapport interne LRI, Paris-XI Orsay, août 1989.*

P. Langley, G. Bradshaw, H Simon : Rediscovering chimistry with the BACON system. *In Machine Learning: An Artificial Intelligence Approach vol 1, R.S. Michalsky, J.G. Carbonnel, T.M. Mitchell Eds, Palo Alto 1984.*

P. Langley, J. Zytkow, H. Simon, G. Bradshaw : Search of regularities : four aspects in scientific discovery, *In* Machine *Learning : An Artificial Intelligence Approach, vol 2, R.S. Michalsky, J.G. Carbonnel, T.M. Mitchell Eds, Palo Alto, 1986.*

D.H. Lebowitz : Experiments with incremental concepts formation, UNIMEM, *in Machine Learning Vol 2, sept 1987.*

I-C Lerman, J. Lebbe, J. Nicolas, P.Peter, R. Vignes : Conceptual clustering in biology : applications and perspectives. *In Data Analysis, Learning symbolic and numeric knowledge, E. Diday ed., ANTIBES 1989.*

M. Manago, Y. Kodratoff : Model Driven Learning of disjunctive concept. *in EWSL 86 Orsay February 1986.*

R.S. Michalski : Variable-Valued Logic and its application to Pattern Recognition and Machine Learning, *in Multiple-Valued Logic and Computer Science, D. Rine ed, North Holland, 1975.*

R.S. Michalski : A theory and methodology for inductive learning. *In Machine Learning : An Artificial Intelligence Approach, (R.S. Michalsky, J.G. Carbonnel, T.M. Mitchell Eds), Palo Alto. 84.*

R.S. Michalski I. Mozetic, J. Hong, N. Lavrac : The AQ15 inductive learning system : an overview and experiments. *Proceedings of IMAL 1986, ORSAY.*

T.M. Mitchell : Generalization as Search. *in Artificial Intelligence N° 18, 1982.*

M-C Perron : Learning differential diagnosis rules from numerical knowledge. *In Data Analysis, Learning symbolic and numeric knowledge, E. Diday ed. ANTIBES 1989.*

J.R.Quinlan : Induction of decision trees. *Machine Learning, Vol 1, pp 81-106, 1986.*

J. Quinqueton, J. Sallantin : Algorithms for learning logical formulas, *in Proceedings IJCAI 1983.*

S. Rajamoney R. DeJong : The classification, detection and handling of imperfect theory problems *in Proceedings IJCAI 87.*

L. Rumelhart, L. Hinton Williams : Learning internal representation by error propagation, *in Distributed Parallel Processing, Vol 1, MIT Press, Cambridge 1986.*

M. Sebag M. Schoenauer : Generation of rules with certainty and confidence factors from incomplete and incoherent learning bases. *Proceedings EKAW 1988, Bonn.*

M. Sebag, M. Schoenauer : Iterative learning and redundant generalizations, *Data Analysis/Learning symbolic and numerical knowledge, E. Diday Ed, NovaScience Antibes 1989.*

C. Vrain, C.-R. Lu : An analogical method to do incremental learning of concepts, *Proceedings EWSL 88, D. Sleeman Ed, Glasgow.*

P.H. Winston : Learning structural descriptions from examples, *in The psychology of Computer Vision, P.H. Winston Eds, McGraw Hill, New York 1975.*

An Incremental Method for Finding Multivariate Splits for Decision Trees

Paul E. Utgoff
Carla E. Brodley
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

Abstract

Decision trees that are limited to testing a single variable at a node are potentially much larger than trees that allow testing multiple variables at a node. This limitation reduces the ability to express concepts succinctly, which renders many classes of concepts difficult or impossible to express. This paper presents the PT2 algorithm, which searches for a multivariate split at each node. Because a univariate test is a special case of a multivariate test, the expressive power of such decision trees is strictly increased. The algorithm is incremental, handles ordered and unordered variables, and estimates missing values.

1 Introduction

For inductive learning, decision-tree methods are attractive for three principal reasons. First, the methods find trees that generalize well to the unobserved instances, assuming that the instances are described in terms of features that are correlated with the target concept. Second, the methods are efficient, generally requiring a total amount of computation that is proportional to the number of observed training instances. Finally, the resulting decision tree provides a representation of the concept that appeals to humans because it renders the classification process self-evident.

This paper presents a new decision-tree algorithm, named PT2, that is designed to provide a richer space of possible tests at a node and to provide a uniform treatment of ordered and unordered variables. Section 2 lays out the issues that motivated the design of PT2, which is presented in Section 3. Following this, Section 4 illustrates the algorithm on three standard learning tasks. Finally, Section 5 draws conclusions about the algorithm and identifies new problems that require further research.

2 Issues for Decision-Tree Induction

This section discusses the principal issues that motivated the design of the PT2 algorithm. These issues arose from attempting to extend the perceptron tree algorithm (Utgoff, 1988) to handle ordered variables, but are central to research on decision-tree induction in general.

2.1 Splitting Criterion

A significant shortcoming of decision-tree algorithms such as ID3 (Quinlan, 1986) is that the space of legal splits at a node is impoverished. A *split* (Morct, 1982) is a partition of the instance space that results from placing a test at a decision node. Each subset of the partition corresponds uniquely to one outcome of applying the test to the instance. ID3 and its descendants only allow testing a single variable (attribute) and branching on the outcome of that test.

In order to facilitate generalization, one would like to avoid a large number of branches at a node. This means that a variable should not have a large number of possible values. This is typically the case for non-numeric variables. However, numeric variables, both continuous and integer-valued, are problematic due to the unlimited range of possible values. A more fundamental distinction among variables is whether a variable's values are ordered or not. Accordingly, an *ordered variable* is one whose possible values are totally ordered. This class includes continuous and integer-valued variables. An *unordered variable* is one whose values are not totally ordered. One standard technique for handling an ordered variable with a large number of possible values is to map its values onto a small set of intervals. Another technique is to partition the values of a numeric variable into two open-ended intervals: those values that are greater than a dynamically determined constant and those that are not (Breiman, Friedman, Olshen & Stone, 1984; Quinlan, 1987).

In general, one would like to allow a richer space of possible splits than those afforded by testing just one variable at a time (Pagallo & Haussler, 1988; Pagallo, 1989). For example, as shown in Figure 1, if the concept to be learned is the set of points in the half-plane $\{(x, y)|y < 2x + 3\}$, then a decision tree based on univariate tests must approximate it with a disjunction of quarter-planes, e.g. $\{(x, y)|(x > -1 \wedge y < 1) \vee (x > 1 \wedge y < 5)\}$. This example illustrates the well known problem that a univariate test can only split a space with a boundary that is orthogonal to that variable's axis. This limits the space of regions in the instance

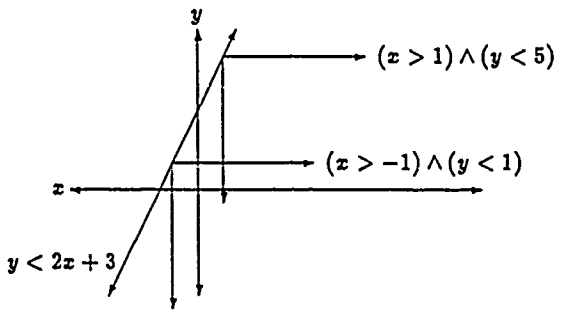


Figure 1: Limitations of Univariate Splits

space that can be represented succinctly, which can result in a large tree and poor generalization to the unobserved instances. The bias imposed by allowing only univariate tests may be inappropriate for a target concept.

2.2 Incremental Tree Revision

For learning tasks in which training instances are provided serially, one would like to be able to revise the existing decision tree, if necessary, instead of rebuilding it from scratch. One does not want a new training instance to render previous learning obsolete. In particular, one would like to be able to conduct the search for a multivariate test incrementally. Several groups, including Qing-Yun and Fu (1983), Breiman et al. (1984), Pagallo and Haussler (1989), Clark and Niblett (1989), and Chan (1989) have devised methods for searching for multivariate tests, but these methods are not incremental.

2.3 Sequential Testing and Understandability

There are two important advantages of decision-tree classifiers that one does not want to lose by permitting multivariate splits. First, the tests in a decision tree are performed sequentially by following the branches of the tree. Thus, only those variables that are required to reach a decision are evaluated. On the assumption that there is some cost in obtaining the value of a variable, it is desirable to test only those variables that are needed. Second, a decision tree provides a clear statement of a sequential decision procedure for determining the classification of an instance. A small tree with simple tests is most appealing because a human can understand it. There is a tradeoff to consider in allowing multivariate tests; simple tests may result in large trees that are difficult to understand, yet multivariate tests may result in small trees with tests that are difficult to understand.

2.4 Mixing Variable Types

One of the well-known problems with decision-tree methods is how to handle ordered variables. As discussed above, such variables are problematic because they may have many possible values, e.g. a continuous variable. For a decision tree, one requires variables with a small number of possible values, so that the number of possible branches at a node is kept small. Techniques exist for mapping an ordered variable to an unordered variable, thereby reducing the number of possible values for the ordered variable from many to as few as two. These techniques are special cases of the more general approach of partitioning an n-dimensional Euclidean space into regions, defined by inequalities. For example, a region in x-y space might be a disc defined by $\{(x, y)|x^2 + y^2 < 3\}$. In general, one can map an n-dimensional point to a region and then treat containment within the region as a two-valued unordered variable.

How can one mix ordered and unordered variables freely? There are two standard approaches. First, as already discussed, one can map each ordered variable to an unordered variable, and then find a Boolean combination that represents the concept. The principal problem with this approach is that Boolean combinations of intervals can only define regions via boundaries that are each orthogonal to one of the coordinate axes. This renders the class of concepts that require other boundary orientations difficult to represent. Alternatively, one can map each variable, ordered or unordered, to a numeric variable and then find a numerical combination that represents the concept. This latter approach allows an accurate and succinct represention for a large class of concepts. For decision-tree induction, one would like to be able to consider boundaries in any orientation and then form regions by splitting the space in terms of these boundaries.

In order to map an unordered variable to a numeric variable, one needs to be careful not to impose an order on the values of the unordered variable. For a two-valued variable, one can simply assign 1 to one value and -1 to the other. If the variable has a range of more than two values, then each variable-value pair can be mapped to a propositional variable, which is TRUE if and only if the variable has the paritucular value in the instance (Hampson & Volper, 1986). This avoids imposing any order on the unordered values of the variable. With this mapping, one can represent concepts over unordered variables, ordered variables, or a mix of such variables. Mapping many-valued variables to two-valued variables has been observed to result in trees with higher classification accuracy (Cheng, Fayyad, Irani & Qian, 1988; Mooney, Shavlik, Towell & Gove, 1989). There are two reasons. First, the two-valued variables provide a finer-grained representation, which makes it possible to find a better decision tree. Cheng et al. point out that this approach also eliminates the irrelevant values of a variable. Second, a binary split of the instance space leaves just two subspaces, placing a larger proportion of the available training instances in each subspace (Pagallo & Haussler, 1988). Because learning in each subspace is based on a larger number of instances, the subtree found will be better determined.

2.5 Missing Values

For some instances, it may be that not all variable values are available. In such a case, one would like to estimate the missing values. This is true for both training and classification. Quinlan (1989) describes a variety of approaches for handling missing values of unordered variables. These include ignoring any instance with a missing value, filling in the most likely value, and combining the results of classification using each possible value according to the probablility of that value.

Another approach for handling a missing value is to estimate it using the sample mean, which is an unbiased estimator of the expected value. For a numeric (ordered) variable, one need only maintain a running total of the values seen and a running count of the number of values seen. This approach can also be applied to missing values of unordered variables. If, as suggested above in Section 2.4, one has mapped every nonnumeric variable to a numeric variable, then this single mechanism also estimates a missing value for a nonnumeric variable. There are two types of mappings to consider. First, if the original variable is two-valued, then it is mapped to a single numeric variable, with one value corresponding to 1 and the other to -1. For a missing value, one simply uses the sample mean. Second, if the original variable is many-valued, then it is mapped to a set of propositional (two-valued) variables, with each one treated as above. For a missing value, one uses the sample mean for each of the propositional variables. One would like to use the sample mean so that the location of the instance in the encoded Euclidian n-space is estimated as accurately as possible. Note that the sample mean of a variable at one node can differ from the sample mean of the same variable at a different node due to the different sets of observed instances on which each is based.

3 The PT2 Decision Tree Algorithm

This section presents the PT2 algorithm for inducing a decision tree from a stream of training instances. The design of the algorithm was motivated by the issues discussed above. The algorithm maintains the decision tree as a global data structure, and revises it incrementally, as necessary, in response to each received training instance. A legal decision tree is either NIL, for the empty tree, or an answer node containing a class name, or a decision node containing one or more Boolean variables, each of which defines a binary split of the instance space. One of these variables is designated as *current*, and has a branch to a decision tree for each of its two possible values. The initial tree is empty.

Table 1 specifies the PT2 algorithm, which is designed to learn single concepts over two classes. At each decision node, it searches for a binary split of the instance space. This search proceeds on two fronts. First, a possible binary split is represented as a linear threshold unit (LTU) over the original input variables. As training instances are observed at an LTU, its weights are adjusted as necessary in order to move its hyperplane in an attempt to separate the positive instances from the negatives. Second, the algorithm attempts to find an LTU at a node that is based on a reduced set of the original input variables. To this end, a set of LTUs is trained at each node in an attempt to identify those variables that can be removed without sacrificing classification accuracy at the node. The rest of this section describes this search in greater detail and discusses the basis on which one split is judged better than another.

Table 1: The PT2 Decision-Tree Update Algorithm.

1. If TREE is empty, then set TREE to new answer node containing class name of training instance, return.
2. If TREE is an answer node, then
   (a) If class name of instance is same as TREE, then return.
   (b) Set TREE to new decision node, initialize current LTU to one based on all n original variables, set $\mathbf{W} \leftarrow 0$, set list of alternate LTUs to empty, initialize other bookkeeping variables.
3. Update decision node at root of TREE, i.e.
   (a) Update the active weight vector $\mathbf{W}$ of every LTU at the node via the absolute error correction procedure. Also update the pocket vector $\mathbf{P}$, and other LTU variables as necessary. If $\mathbf{P}$ of current LTU has changed, then discard its subtrees if they exist. Use the sample mean for any missing value. Whenever $\mathbf{P}$ changes, also save the sample means that correspond to $\mathbf{P}$.
   (b) If the current LTU is not yet determined (see text), then return.
   (c) If some alternate LTU is determined and is at least as good (see Table 2) as the current LTU, then
      i. Replace the current LTU with a best such alternate LTU.
      ii. Remove from the current LTU all original variables for which all the associated weights of the encoded variables are 0.
      iii. Reset the list of alternate LTUs to those based on all $n - 1$ variable combinations of those in the new current LTU. For each alternate LTU, initialize its $\mathbf{W}$ by setting each w_i to the corresponding p_i of the new current LTU, update the LTU via the absolute error correction procedure. Use sample mean for any missing value.
   (d) Descend recursively along branch below $\mathbf{P}$ of current LTU as per instance, return.

3.1 Training an LTU

A linear threshold unit can be used as a Boolean variable, specifically as a test at a decision node, because it is a predicate over its inputs. The LTU maps its variables to either the positive or negative side of its

hyperplane. All original variables are encoded as necessary, as described in Section 2.4, to achieve a set of numeric variables, called the encoded variables. These are normalized dynamically so that the maximum observed value a of a variable maps to 0.5 and the minimum observed value maps to -0.5. The variables are normalized so that each has the same influence during the error computation for the absolute error correction rule. The mapping $M(a)$ is a function of the historical minimum and maximum observed for the variable. Let

a = value of the encoded variable
$hmin$ = historical minimum for variable
$hmax$ = historical maximum for variable

Then

$$M(a) = \begin{cases} -0.5 & \text{if } hmax = hmin \\ \frac{a - hmin}{hmax - hmin} - 0.5 & \text{otherwise} \end{cases}$$

As training instances are observed, each LTU is adjusted as necessary via the absolute error correction procedure (Nilsson, 1965). Combining input features to form multivariate splits is a form of constructive induction; new terms are created based on linear combinations of subsets of the original variables.

3.2 Eliminating Variables

If, at a node, the designated current LTU based on n original variables is no better than one of the alternate LTUs, each based on $n-1$ of these variables, then the set of LTUs being considered at the node is changed. A best such alternate LTU is designated as current, the others are discarded, and a new set of alternate LTUs is created, based on leaving out one variable from those of the new current LTU. This is done in order to find a test on fewer variables if possible. The idea of removing one original variable at a time is taken from CART (Breiman, Friedman, Olshen & Stone, 1984), and is based on the assumption that it is better to search for a useful projection onto fewer dimensions from a relatively well informed state than it is to search for a projection onto more dimensions from a relatively uninformed state. Note that when one original variable is removed, one or more encoded variables are removed.

3.3 Comparing Splits

How does one decide whether one split, as manifested by an LTU, is better than another, so that a best LTU can be designated current? This is accomplished with a procedure that depends on Gallant's (1986) Pocket Algorithm. For a linear threshold unit, the algorithm saves in **P** the best weight vector **W** that occurs during normal perceptron training, as measured by the longest run of consecutive correct classifications, called the *pocket count*, assuming that the observed instances are chosen in a random order. Gallant shows that the probability of an LTU based on the pocket vector **P** being optimal approaches 1 as training proceeds. The pocket vector is optimal in the sense that no other weight vector visited so far is likely to be a more accurate classifier. The Pocket Algorithm fulfills a critical role when searching for a separating hyperplane because the classification accuracy of the LTU based on **W** is unpredictable when the instances are not linearly separable (Duda & Hart, 1973).

Table 2: Procedure to Determine Whether LTU1 is Better than LTU2.

1. If either of LTU1 or LTU2 is undetermined then return FALSE.
2. If the pocket count of LTU1 is higher than that of LTU2, then return TRUE.
3. If the pocket count of LTU1 is identical to that of LTU2, and LTU1 is based on fewer original variables, then return TRUE;
4. Return FALSE.

One might simply assume that the LTU with the highest pocket count provides the best split, but there are two problems with such an assumption. The procedure shown in Table 2 was devised to decide whether one LTU provides a better split than another, and it depends on the definition of *determined* given below. To understand the need for this procedure, consider the problems that would arise from using the pocket count alone.

First, one cannot select an LTU if its pocket vector fails to discriminate among the instances. Consider a learning problem in which the training instances belong predominantly to one class. An LTU can classify a long sequence of instances correctly if it always classifies each one as the more frequently occurring class. Indeed, such a split may result in higher classification accuracy than any split that actually discriminates instances in one class from the other. However, a split that does not discriminate is no split at all. If the space of instances at a node is not split, then the space of instances at one subtree will be identical to the space at the parent. Thus, the same null split would be found at the subtree, the process would repeat, and the tree would become infinitely deep. To avoid this, an LTU with a pocket vector that is not based on having observed instances from more than one class can never be considered better than another LTU.

The second problem is that if the current LTU is not a perfect classifier, then subtrees will be needed to split the space further. It can be wasteful to try to grow subtrees before there is any strong indication that a given LTU is the best that can be found at the node. This is because the algorithms calls for discarding both subtrees of a node whenever the pocket vector **P** of the current LTU is redefined. Such activity will eventually cease because the pocket count of the current LTU increases monotonically. However, to reduce lost effort, no subtrees are allowed to grow below an LTU until enough evidence has accumulated to indicate that an improved pocket vector is not apt to be found anytime soon. The status of an LTU is considered to be *determined* if and only if the following

conditions are true:

1. At least kn, $k \geq 2$, instances must have been presented to the LTU, where n is the number of original variables. This is a minimal test based on the capacity of a hyperplane (Duda & Hart, 1973); if fewer than $2n$ instances have been observed, then the LTU is known to be underdetermined.
2. As discussed above, the pocket vector for the LTU must separate at least one instance from each class.
3. Either; the pocket vector **P** is identical to the active weight vector **W**, or each and every weight in **W** has been varying within its historical minimum and maximum for a number of weight adjustments greater than the log of the number of weights in **W** (Utgoff, 1988; Utgoff, in press).

Note that the status of the LTU can change back and forth between determined and undetermined if a new historical minimum or maximum is established infrequently. Also note that one can raise k to cause the algorithm to be more conservative about determining the status of an LTU. In the current implementation of PT2, the default value of k is 5.

4 Illustrations

This section illustrates the PT2 algorithm on three standard learning tasks. The first is the DNF concept used to illustrate the FRINGE algorithm (Pagallo, 1989). This concept is expressed succinctly by allowing multivariate tests at a node. Second is the multiplexor, both for the six-bit and three-bit cases. Decision-tree algorithms normally do very poorly on this problem because the best variable to test at the root is not correlated with the classification. The final illustration is Quinlan's (1987) hyperthyroid concept, in which instances are described by a mix of unordered and ordered (numeric) attributes, some with missing values.

4.1 The DNF Task

The DNF task was used previously to illustrate the ability of FRINGE to find multivariate tests at a node, though by a much different mechanism from that of PT2. The concept to be learned is the Boolean function $ab \vee c\bar{d}e$. Although there are only 32 possible instances, the algorithm always obtains its next training instance by selecting randomly from the full space of 32 instances. Figure 2 shows the tree found by PT2 after training on 775 instances. For clarity, the weights have been scaled by a constant factor. The tree consists of two tests and three leaves, and is logically equivalent to that found by FRINGE. Recall that PT2 automatically encodes TRUE as 0.5 and FALSE as -0.5. The tree found by ID3 for this task consists of eight tests and nine leaves.

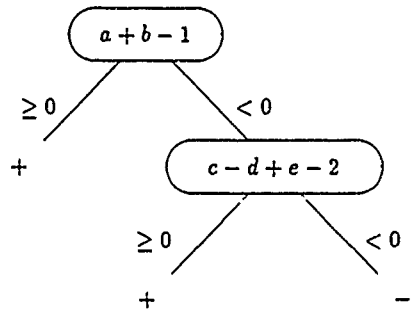


Figure 2: Tree for the DNF Task

4.2 The Multiplexor Task

PT2 was run on both the six-bit and the three-bit multiplexor tasks (Barto, 1985; Wilson, 1987; Quinlan, 1988). The three-bit multiplexor is of interest for the sake of illustration rather than the complexity of the task. The three-bit multiplexor is a Boolean function over three Boolean variables. One variable, here called f, is called the address bit, and it serves as a selector of one of the two variables g, and h, called the data bits. The value of the function is the value of the selected data bit. The simplest tree for this concept corresponds to the expression $fg \vee \bar{f}h$. The problem is interesting because the address bit is the best test at the root, yet it is not correlated with the classification, although the data bits are. Thus, for algorithms that choose a split on such a basis, any data bit looks like a better choice than any address bit. There exists a split on all three variables that is correct for 7 of the 8 instances, and PT2 finds it, as the pocket count is highest for this split. PT2 found a correct tree after training on 480 instances, as shown in Figure 3. This PT2 tree contains two tests and three leaves, whereas the tree found by ID3 contains five tests and six leaves respectively. Note that the test at the root evaluates all three variables, which is also suboptimal. One would prefer a tree in which only necessary tests are performed.

For the six-bit multiplexor task PT2 found a correct tree of ten tests and eleven leaves after training on 3,968 instances from the full space of 64 possible instances. Total CPU time during training was 143 seconds. A characteristic of the PT2 algorithm is that the first correct tree that it finds may not be the smallest that it would find if training were to continue. It is possible that the current LTU at a node will be replaced by one that is better. Such improvements can result either from an improved pocket vector in the current LTU, or by replacing the current LTU with one that is based on fewer of the original input variables. For the six-bit multiplexor task, PT2 was left to continue training until it had seen 71,800 training instances. Although the final tree was the same size as the first correct tree, in terms of tests and leaves, some of the LTUs were replaced by LTUs based on fewer variables. The total number of variables in all ten LTUs of the first correct tree was 38, but this total in the final tree was 25. The tree that ID3 produces for this task contains 25 tests and 26 leaves.

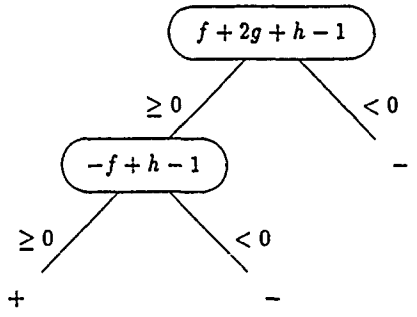
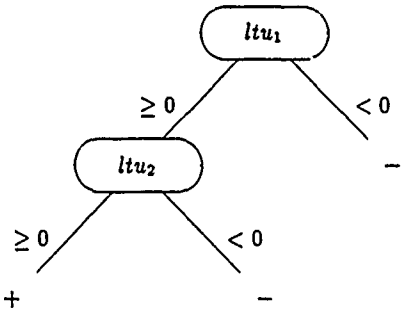


Figure 3: Tree for the Three-Bit Multiplexor Task



$$
\begin{aligned}
ltu_1 = {} & 61.6 ftival + 20.9 t3val - 20.5 onthy + \\
& 15.4 tt4val + 8.5 tt4 - 7.3 t4uval + \\
& 6.5 tbg + 6.5 hpit - 6.5 psyc + \\
& 5.5 ref.other - 5.5 thsurg + 5.0 ref.svi + \\
& 4.5 preg + 4.5 ref.svhc - 4.5 qhypo - \\
& 4.5 qhyper - 3.9 thsval + 3.5 ref.stmw + \\
& 3.5 lith - 3.5 onanti - 3.5 sick + \\
& 0.5 goit + 0.5 ths - 0.5 qonthy - \\
& 0.5 i131 + 6.5
\end{aligned}
$$

$$
\begin{aligned}
ltu_2 = {} & -38.7 thsval + 20.9 t4uval + 13.1 ftival - \\
& 11.4 tt4val + 4.5 sick + 4.5 preg + \\
& 4.5 i131 + 4.5 qhypo + 4.5 t4u + \\
& 4.5 fti - 3.5 ths + 3.0 ref.svhc - \\
& 1.5 qonthy - 1.5 ref.other + 0.6 age + \\
& 0.5 tbg + 0.5 t3val + 0.5 tt4 + \\
& 0.5 thsurg + 0.5 onthy + 0.5 onanti + \\
& 0.5 lith + 0.5 goit + 0.5 hpit + \\
& 0.5 psyc - 0.5 ref.svi - 0.5 tum + \\
& 0.5
\end{aligned}
$$

Figure 4: Tree for the Hyperthyroid Task

4.3 The Hyperthyroid Task

The hyperthyroid task (Quinlan, 1987) is of interest because its instances are described in terms of both numeric and nonnumeric variables, and because there are training instances with missing values for both types of variables. The training set contains 2,800 instances, each falling into one of four classes. Because PT2 currently handles only two-class problems, the task was cast as learning the concept of "hyperthyroid". Instances labelled as "hyperthyroid" were considered to be positive, and all other instances were considered to be negative. Each instance is described by 28 variables (one variable whose value is missing for all the instances is not included). Across all 28 variables for the 2800 training instances, there are 1,756 missing values, for an average of 0.63 missing values per instance. The independent test set contains 972 instances described in terms of the same variables, with a total of 536 missing values, for an average of 0.55 missing values per instance.

Due to the lopsided distribution of the training instances, with 97.79% being negative, a variation of random sampling was used for selecting the next instance during training. Instead of sampling randomly from the entire population of training instances, the instances from the two classes were kept as two separate populations. An instance was selected randomly within a population but the choice of population alternated each time. This training strategy supplies an even distribution of instances at the root, but the effect diminishes at the subtrees because the tree is attempting to separate the positives from the negatives. PT2 does not depend on this training strategy. The rationale is simply that with a lopsided distribution, most instances will come from one class and therefore be uninformative.

Figure 4 shows the tree found by PT2, having trained on a total of 372,400 instances for approximately one clock week. The tree classifies 99.46% of the training data correctly and 99.07% of the test data correctly. In the figure, the notation $a.v$ indicates a propositional variable corresponding to value v of the original input variable a. Thus, ltu_1 contains 25 weights based on 22 original variables, and ltu_2 contains 27 weights based on 25 original variables.

It is unclear whether 99.07% correct classification on the test data is good, especially given that simply always guessing negative yields 98.25% correctly classified. Precise classification accuracies of previous solutions related to this task (Quinlan, 1987; Chan, 1989) were not published. It is worth noting that both C4 and PT2 chose *ftival* as the most important variable at the root. Also note that for PT2, one can rank the relative importance of the variables by comparing the magnitudes of their respective weights. It is meaningful to compare weights because PT2 normalizes all the variable values.

5 Discussion

This section identifies strengths and weaknesses of the PT2 algorithm, and indicates the directions in which the research is proceeding.

5.1 Strengths

There are four strengths of the algorithm. First, it is incremental, which means that additional training instances received at a later time will not obviate previous learning by requiring that a new tree be built from scratch. This is highly desirable for learning algorithms that are imbedded in systems that learn from experience.

Second, the algorithm finds a multivariate test at a node by training a linear threshold unit. This allows defining regions in the instance space that have boundaries in any orientation. This richer space of splits, compared to allowing only univariate splits, enables the program to find more compact trees. The algorithm attempts to find a linear split that is optimal in terms of classification, but that is based on a small subset of the original variables.

Third, the algorithm treats all variable types uniformly by encoding their ranges numerically. The encoded values are normalized by mapping each range onto the interval [-0.5,0.5]. Encoding and mapping are determined dynamically. This encoding allows one to learn concepts over instances that are described by a mix of ordered and unordered variables, rather than one or the other.

Finally, the algorithm estimates missing values via sample mean, which is an unbiased estimator of the expected value. This is well defined for all variable types because they are all encoded numerically.

5.2 Weaknesses

There are four clear weaknesses in the algorithm, which are being addressed as this work continues. First, the mechanism for dropping variables from a LTU is too costly. The algorithm may train $O(n^2)$ LTUs at a node while searching for a good split. This is worse than the predecessor perceptron tree algorithm (Utgoff, 1988), which trained just one LTU at each node.

Second, there is no guarantee that PT2 will find a tree that satisfies the sequential testing and understandability goals outlined in Section 2.3. Indeed, the tree found for the hyperthyroid task is poor in this regard.

Third, the algorithm does not take advantage of pruning techniques that have been designed to prevent overfitting the training data (Breiman, Friedman, Olshen & Stone, 1984; Mingers, 1989). Furthermore, the algorithm is also subject to underfitting error. If too few unique instances are delivered to a node, with respect to the dimensionality of the instance descriptions, then an LTU will be underdetermined (Duda & Hart, 1973).

Finally, the algorithm is limited to discriminating just two classes.

5.3 Near-Term Extensions

The algorithm is being extended in three ways. First, a better method for discarding variables at a node is being investigated. Given that the encoded variables are normalized, one can identify the most important variables by the magnitudes of the weights. One could base the LTU for the split on just those variables that have sufficiently large magnitudes, compared to the largest magnitude among the weights..

Second, the ability to discriminate more than two classes is being added. Several known methods are being considered, but the most promising is to separate the instances into two superclasses at each node (Breiman, Friedman, Olshen & Stone, 1984). Such a scheme finds "strategic" splits, in the sense that splits near the top of the tree group together those classes that are similiar, while splits near the leaves isolate single classes.

Third, a pruning mechanism is being added that will attempt to prevent overfitting and underfitting problems.

5.4 Longer-Term Extension

A longer term problem that may be addressed is how to detect when the piece-wise linear classification strategy is performing poorly and alter it automatically. For example, if the members of the target concept define a hyperregion with one or more curved boundaries, then a potentially infinite number of hyperplanes will be needed to approximate the hyperregion. This would lead to a very large tree, yet if the space were to be split with hyperspheres or hyperellipses then the tree would be smaller, which would facilitate learning.

Acknowledgments

This material is based upon work supported by the National Aeronautics and Space Administration under Grant No. NCC 2-658, and by the Office of Naval Research through a University Research Initiative Program, under contract number N00014-86-K-0764. Sharad Saxena, Jamie Callan, Tom Fawcett, David Haines, David Lewis, Jeff Clouse, Margie Connell, and Pat Langley provided helpful comments.

References

Barto, A. G. (1985). Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology, 4*, 229-256.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth International Group.

Chan, P. K. (1989). Inductive learning with BCT. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 104-108). Ithaca, NY: Morgan Kaufmann.

Cheng, J., Fayyad, U. M., Irani, K. B., & Qian, Z. (1988). Improved decision trees: A generalized version of ID3. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 100-106). Ann Arbor, MI: Morgan Kaufman.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning, 3*, 261-283.

Duda, R. O., & Hart, P. E. (1973). *Pattern Classification and Scene Analysis.* New York: Wiley & Sons.

Gallant, S. I. (1986). Optimal linear discriminants. *Proceedings of the International Conference on Pattern Recognition* (pp. 849-852). IEEE Computer Society Press.

Hampson, S. E., & Volper, D. J. (1986). Linear function neurons: Structure and training. *Biological Cybernetics, 53*, 203-217.

Mingers, J. (1989). An empirical comparison of pruning methods for decision tree induction. *Machine Learning, 4*, 227-243.

Mooney, R., Shavlik, J., Towell, G., & Gove, A. (1989). An experimental comparison of symbolic and connectionist learning algorithms. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 775-780). Detroit, Michigan: Morgan Kaufmann.

Moret, B. M. E. (1982). Decision trees and diagrams. *Computing Surveys, 14*, 593-623.

Nilsson, N. J. (1965). *Learning Machines.* New York: McGraw-Hill.

Pagallo, G., & Haussler, D. (1988). *Feature discovery in empirical learning* (unpublished). Santa Cruz, CA: University of California, Department of Computer and Information Science.

Pagallo, G. (1989). Learning DNF by decision trees. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 639-644). Detroit, Michigan: Morgan Kaufmann.

Qing-Yun, S., & Fu, K. S. (1983). A method for the design of binary tree classifiers. *Pattern Recognition, 16*, 593-603.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81-106.

Quinlan, J. R. (1987). Decision trees as probabilistic classifiers. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 31-37). Irvine, CA: Morgan Kaufmann.

Quinlan, J. R. (1988). An empirical comparison of genetic and decision-tree classifiers. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 135-141). Ann Arbor, MI: Morgan Kaufman.

Quinlan, J. R. (1989). Unknown attribute values in induction. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 164-168). Ithaca, NY: Morgan Kaufmann.

Utgoff, P. E. (1988). Perceptron trees: A case study in hybrid concept representations. *Proceedings of the Seventh National Conference on Artificial Intelligence* (pp. 601-606). Saint Paul, MN: Morgan Kaufmann.

Utgoff, P. E. (in press). Perceptron trees: A case study in hybrid concept representations. *Connection Science, 1.*

Wilson, S. W. (1987). Classifier systems and the animat problem. *Machine Learning, 2*, 199-228.

Incremental Induction of Topologically Minimal Trees

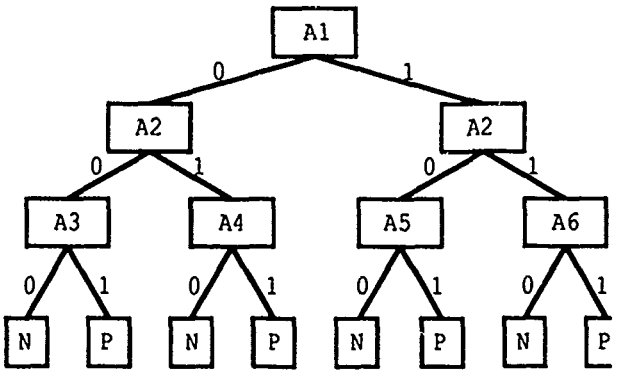
Walter Van de Velde
Centre d'Estudis Avançats de Blanes
Camí de Santa Bàrbara s/n, E-17300 Blanes, Spain
Email: walter@ceab.es

Abstract

This paper proposes the notion of 'topological relevance' as a means to formalize the complexity of a decision tree representation of a set of instances. It then presents an incremental algorithm, called IDL, for the induction of decision trees which are optimal according to this notion. IDL relies on ideas from ID4 and ID5 but searches using a statistical criterion for expanding nodes and a tree topological one, called topological relevance gain, for transforming decision trees. The results of its analysis and empirical validation show that, with provisions, IDL rapidly finds the same or a better tree than top-down induction algorithms and their incremental versions ID4, ID5 and ID5R.

1. Introduction

What is a good decision tree? Despite the fact that induction of decision trees is one of the best developed subfields of machine learning (see e.g. [2,3,5,9]), this question has not really been answered. Of course 'goodness' has multiple facets, some of them depending on the application, but classification accuracy and tree complexity are important factors. But could one recognize a good tree if one sees one? Our quality measures are too vague to do so. It is a general shortcoming of top-down induction of decision tree algorithms (TDIDT, [5]) that there is no intentional description of the tree they construct. Starting from a set of examples, usually described as a set of attribute-value pairs annotated with a class TDIDT algorithms successively split the set using a good (e.g. most informative) attribute until sets of examples in a single class are left. The splits become the nodes in the tree and the leaves are labeled with the class of the examples. The tree such an algorithm is looking for is simply the one it happens to come up with. There is no reason why this should be in some sense the best one. Accuracy of the trees tends to vary little across attribute selection measures [2]. However, size does vary and is often much larger than optimal. For example the following is one of the two smallest decision trees for the 6-multiplexer, and far beyond the capabilities of TDIDT algorithms ([6], see also figure 2):



In this paper I precisely define a notion of topological minimality of decision trees. The above tree is an example of a topologically minimal tree. Intuitively speaking, a decision tree which correctly classifies a set of examples is topologically minimal if an attribute is localized as much as possible in the tree avoiding, for example, obsolete replications of subtrees. TDIDT algorithms may find these topologically minimal trees but in other cases, like for multiplexer concepts, fail to do so. So I go on to present an algorithm, called IDL, which was designed to incrementally construct such optimal trees. IDL searches using a statistical selection measure like any of those TDIDT would use [2] for expanding nodes, but a new and topological measure, called *topological relevance gain*, for transforming a tree into a smaller one. Topological relevance is measured by using the decision tree in a bottom-up (hypothesis-driven) fashion. It expresses the import of an attribute for classifying an example based on tree topology and not, as is usual, on statistics over a training set. Formal analysis is preliminary but experiments show that IDL is good at finding a topologically minimal tree if it exists, and with less work than the other incremental algorithms ID4 [7], ID5 [9] and ID5R [11] which often find suboptimal trees. Current IDL deals badly with noise but I will ignore that problem in this paper.

The paper is structured as follows. Section 2 motivates and defines the notions of topological relevance and topological minimality. Section 3 presents the IDL algorithm and a detailed example. Section 4 presents the results of complexity analysis and some experiments comparing IDL with ID5R. Section 5 describes related and future work and some open problems.

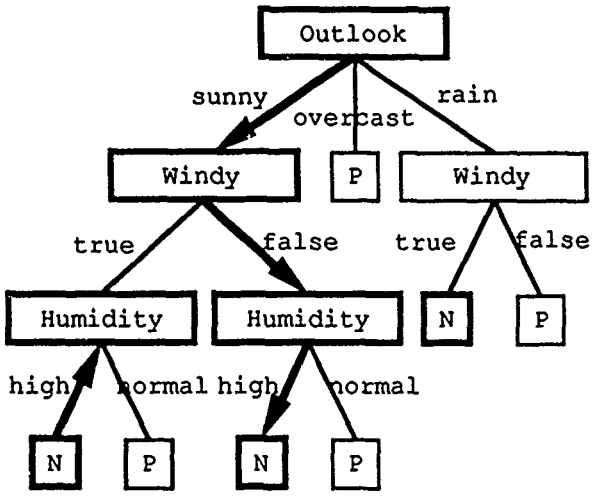
2. Topological Relevance

Given a tree T with root N, what can be said about the relevance of an attribute A for classifying an example E? I propose to consider a topological notion of relevance (i.e. based on the structure of the tree) instead of a statistical one (i.e. based on statistics over the training set). I will use the example domain from [5] to illustrate the idea. The set of examples is shown in table 1.

Table 1: The Weather Concept from [5]

	Attributes				
	Outlook	Temperature	Humidity	Windy	
1	sunny	hot	high	false	N
2	sunny	hot	high	true	N
3	overcast	hot	high	false	P
4	rain	mild	high	false	P
5	rain	cool	normal	false	P
6	rain	cool	normal	true	N
7	overcast	cool	normal	true	P
8	sunny	mild	high	false	N
9	sunny	cool	normal	false	P
10	rain	mild	normal	false	P
11	sunny	mild	normal	true	P
12	overcast	mild	high	true	P
13	overcast	hot	normal	false	P
14	rain	mild	high	true	N

Assume one has the following tree T:



Consider example 1 (call it E). This example is classified correctly. The branches and the leaf node involved in the classification process starting from the root of the tree form the *classification path* of E:

(outlook=sunny windy=false humidity=high N)

This classification path is highlighted with arrows pointing downwards (i.e. the direction of the classification process). I say that *a path covers an example* if all tests on it are satisfied by that example, including the class decision. Here a branch is interpreted as a test whether the test attribute at the parent node has as value the label of the branch. The classification path obviously covers the example and moreover it is the only complete one through T, i.e. one connecting the root with a leaf. In addition, there are a number of partial paths that also cover the example. Some of these are found by using the tree in a bottom-up fashion: Starting from all leaves in the tree which are labelled with N, the example climbs the branches who's test it satisfies. In this way the classification path and a number of partial paths covering the example are reconstructed. These partial paths have been highlighted in the previous figure with arrows pointing upwards (i.e. the direction in which they are constructed). There are two such partial paths:

(N) and (humidity=high N)

The occurrence of the attribute humidity on the classification path as well as on one of the partial paths can be interpreted as follows:

(1) it confirms the relevance of humidity for the class-membership of E because it plays a role in the hypothesis-driven confirmation of E's classification result (N). Judged by tree structure, humidity=high is highly predictive for the class N.

(2) the value of the closest common attribute above the two humidity-nodes (windy) does not influence the role of humidity in E's classification. Branching on windy does not change the way in which humidity is used.

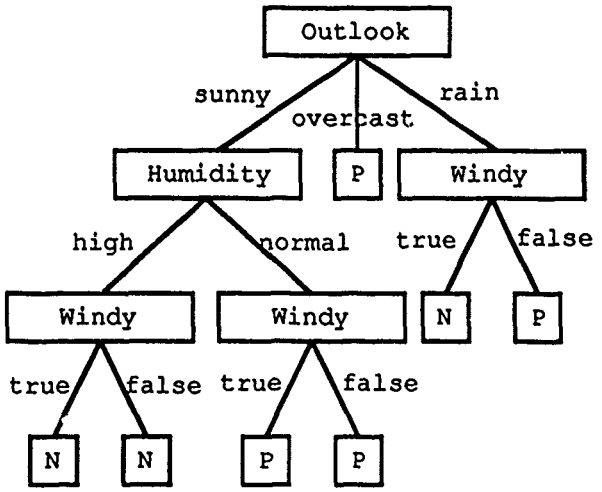
For a given tree T, attribute A and example E in class C, let us call the number of occurrences of A on the (complete or partial) paths reconstructed by hypothesis-driven tree climbing using E and starting from T's leafs labeled C, the *topological relevance* $TR_T(A,E)$ of A for E. Thus, these are the scores for topological relevance:

$TR_T(outlook,E)=1$
$TR_T(temperature,E)=0$
$TR_T(humidity,E)=2$
$TR_T(windy,E)=1$

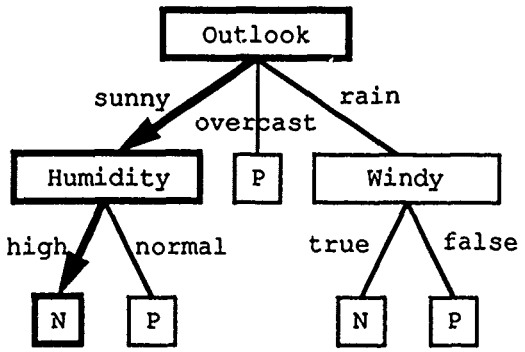
The first observation above suggests that topological relevance measures the import of an attribute for determining the class of an example (i.e. the predictive power of the attribute-value for the class). The second observation supports the hypothesis that testing humidity first may make the windy test obsolete. Therefore, if both attributes can be switched chances are that the tree (actually its subtrees) can be pruned.

A tree transformation technique which is also used in ID5 and ID5R [9, 11] can be used to swap the attributes windy and humidity. This technique, which is justified by commutativity of attribute tests, interchanges the windy and humidity attributes and regroups the subtrees

at the second level down (in this case the four leaves). One obtains the following tree:



The windy tests have now become obsolete and pruning at these nodes results in the following decision tree:



The classification path of E is highlighted in the pruned tree. The scores for the topological relevances of the attributes for the example in this tree are now the following:

TR_T(outlook,E)=1
TR_T(temperature,E)=0
TR_T(humidity,E)=1
TR_T(windy,E)=0

The topological relevance of the attribute humidity is now 1, which is as low as it can get if humidity is relevant at all. So in this case the transformation did minimize topological relevance, intuitively meaning that an attribute's activity is localized as much as possible in the tree (compare with the topological redundancy in the original tree). This was achieved by pulling up topologically more relevant attributes in the tree. This is the intuitive basis of the IDL algorithm which is described in section 3.

Note that topological relevance of an attribute for an example is not the same as the number of occurrences of an attribute in the tree. For example, in the original tree the topological relevance TR_T(windy,E) is 1 despite the apparent symmetry of the tree with respect to that attribute. Also note that the topological relevance of an attribute is at least one for an attribute which is used on the classification path.

It is easily verified that for the final tree the topological relevance of all attributes for all examples in table 1 is equal to 0 or 1. This means that there is no topological redundancy in the tree: there is no replication of decision nodes with a similar role for classification. Similar partial classification paths are represented by the same branches.

I say that a tree T is *topologically minimal* with respect to a set of examples $\{E_i\}_i$, if it is correct, cannot be pruned without loss of correctness and $TR_T(A,E_i)\leq 1$ for each attribute A and every i. (I also assume that a decision tree is well-formed meaning that it has no decision nodes with only one branch (infra)). So the final decision tree shown above is topologically minimal with respect to the training set from table 1. The optimal tree for the 6-multiplexer which was shown before is also topologically minimal in the above sense with respect to the 64 instances.

3. IDL

IDL is a new incremental algorithm for the induction of decision trees which are topologically minimal in the sense explained in section 2. Earlier incremental decision tree algorithms, in particular ID4 [7], ID5 [9] and ID5R [11] are essentially incremental versions of TDIDT [5]: they use backtracking (in ID4) or simulated backtracking (in ID5 and ID5R) to recover as best as possible and with minimal loss of training effort from deviations from TDIDT's search path (general to specific hill-climbing without backtracking) which is considered to be "ideal" and leading to the best tree. IDL builds on this previous work and uses two ideas which stem from ID4 and ID5 respectively:

ID4: one can maintain attribute-value and class counts for every potential test attribute at every node and use these to find the best test attribute for splitting according to a statistical selection measure [2] without re-examining the examples;

ID5: a test attribute can be replaced by another one by a tree transformation technique which pulls up an attribute from below without re-examining the part of the examples which is implicit in the decision tree. Moreover attribute-value and class counts are easily recalculated.

IDL performs a heuristic search using three types of search steps, namely *specialization* by splitting, *generalization* by pruning, and *transformation* by ID5's pull-up technique [9,11]. IDL's increased power stems from two insights:

Table 2: The IDL Tree Update Algorithm

Let T be the current tree and E the next example. To update the tree T using the example E do the following:

1.Classify

Classify E, updating the count information at each node on the classification path. Let N be the node where E ends up (i.e. a leaf or a split node with a missing branch).

2. Expand

2.1. If E is incorrectly classified then expand N into a tree using TDIDT and the statistical criterion.

2.2. If E cannot be classified then add a new branch to N and label its leaf with the class of E.

2.3. Classify E in the subtree rooted at N (updating count information) and remember the part of E which is not implicit in the classification path at the leaf L where E ends up in.

3. Restructure

Recursively update the test attribute of the nodes on the entire classification path of E from leaf L to root T.

1. search using these transformation steps can potentially come up with better trees than TDIDT because it lets hypotheses be reached in parts of the search space which are remote from TDIDT's search path;
2. a measure based on *topological relevance* (see section 2) can be used for selecting transformations. It exploits the knowledge from the structure of the tree before transformation to create opportunities for pruning after transformation.

Thus, instead of using transformations to stay close to TDIDT's search path, IDL uses transformations to come up with better trees. It therefore uses two different selection measures: a statistical one like any of those TDIDT would use for splitting (specialization) and the topological one for selecting transformations. IDL prunes (generalization) wherever it finds obsolete splits.

3.1. IDL: Algorithm

IDL starts with an empty tree and processes examples one by one. With the first example IDL creates a root-node and labels it with the class of the example. Assume a fixed statistical criterion for splitting nodes (e.g. any of those from [2]). The basic algorithm for processing an example using the current tree is shown in table 2. This skeleton is very similar to ID5's, except that ID5 updates test attributes from root to leaf.

IDL also uses the transformation technique from ID5(R) [9,11] to pull up an attribute to a node. This technique is based on an operation which switches the levels of attributes in the tree while preserving (and possibly increasing) classification accuracy. In IDL the pull-up technique requires one extra operation. Attribute switching regroups subtrees and there is no reason why this should result in groups with more than one subtree (see [9], [11] for details). In ID5(R) the resulting single

Table 3: The IDL Attribute Revision Algorithm

Let M be a node on the classification path of E. Let $A_1 \ldots A_n$ be the attributes used on this path down from M and excluding the test attribute at M.

3.1. For each leaf under M which is labelled with the same class a E, reconstruct the largest path starting from this leaf and covering the example E, using only attributes among $A_1 \ldots A_n$.

3.2. For each attribute $A_1 \ldots A_n$ compute its topological relevance $TR_M(A_i,E)$, which is simply the number of occurrences on all the reconstructed paths. Note that $TR_M(A_i,E) > 0$.

3.3. For each attribute $A_1 \ldots A_n$ also compute the topological relevance $TR_S(A_i,E)$ where S is the immediate son of M on the classification path of E. Note that $TR_M(A_i,E) \geq TR_S(A_i,E)$

3.4. For each attribute $A_1 \ldots A_n$, compute the *topological relevance gain* as the weighted increase in topological relevance between S and N, i.e.:

$$TRG_M(A_i,E) = (TR_M(A_i,E) - TR_S(A_i,E)) / TR_M(A_i,E)$$

3.5. If some attribute among $A_1 \ldots A_n$ has a positive topological relevance gain then pull up the one with the highest score to M. Ties are resolved by choosing the attribute nearest to M.

3.6. Otherwise compute the best test attribute A_{best} at M according to the statistical criterion and assure its presence in the tree with root M. To assure the presence of A_{best} in the tree with root M, do nothing if A_{best} is already used in this tree. Otherwise, pull up A_{best} to M. (The pull-up process will make A_{best} appear).

3.7. Prune any subtree of M which is obsolete, i.e. all leaves under it are labeled with the same class.

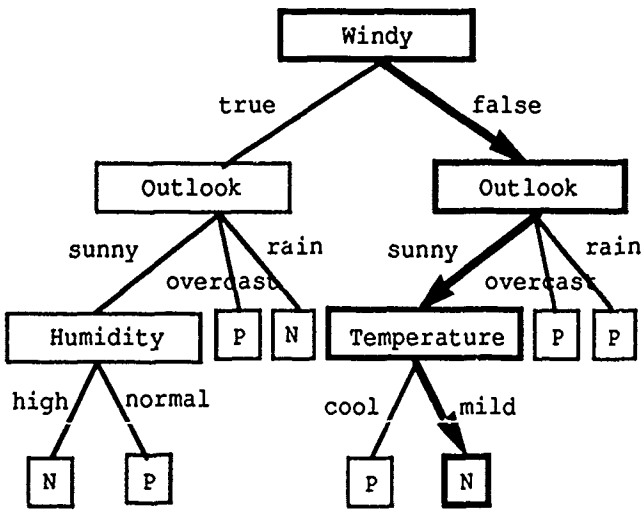
branch splits, being uninformative, will automatically disappear through subsequent training (in ID5) or recursive restructuring of the subtrees (in ID5R). In IDL this is not the case and they need to be explicitly removed. This is achieved by applying the same pull-up technique to push the uninformative attribute down and eventually out of the tree (see also [12]).

The crucial distinction between IDL and the other algorithms is in the heuristic for applying tree transformations. The algorithm to select a better test attribute is shown in table 3. IDL uses the example to guide its search for useful transformations in the most relevant parts of the tree. The role of step 3.6 is to assure enough diversity among the attributes. When a tree has become fully accurate there will be no more expansions of nodes (step 2) and the transformations (step 3.5) alone will never lead to the appearance of a new and possibly crucial attribute. As in ID5(R) it is useful to keep pruned subtrees around (virtual pruning) because they embody training effort which may be relevant for subsequent pull-up operations.

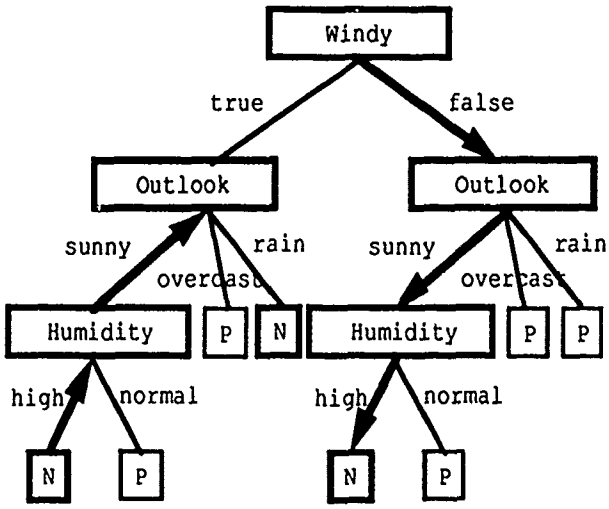
Note that IDL may find the same tree as the other algorithms. It does so however mostly based on topological considerations instead of on statistical ones. I refer to figure 2 for a case in which this makes a substantial difference.

3.2. IDL: Example

The activity of IDL in the early stages of training consists mostly of node expansions (step 2) so that the tree becomes larger and more accurate. The larger the tree the higher the chance for topological redundancy. Suppose one has reached this tree which correctly classifies all but one example (example 1 cannot be classified):

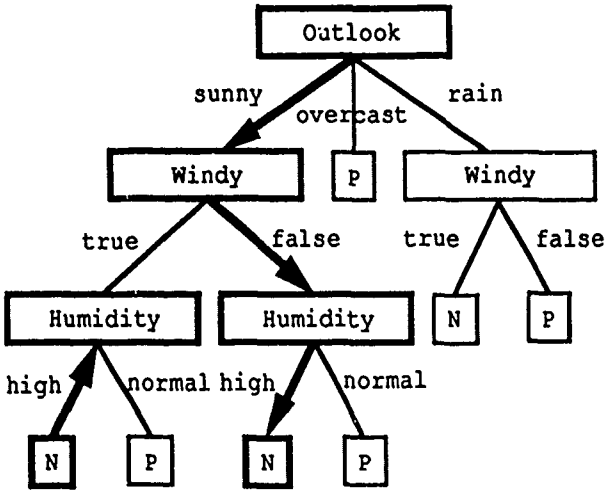


Suppose example 8 is next. It is classified correctly (step 1) and IDL continues by revising the test attributes on the classification path, from leaf to root (step 3). At the temperature node no topological relevance gains are computed (step 3.5 fails). Suppose that the statistically most relevant attribute at that node is humidity. It is not present in the subtree (nothing is) so it is pulled up (step 3.6) by expanding two leaves using humidity, and swapping humidity with temperature. Subsequent pruning removes the obsolete temperature splits (step 3.7). After revising one node the tree is as follows:



The classification path of example 8 in the new tree is highlighted. IDL now revises attributes higher up on this path. At the outlook node this generates no action. At the root node the paths are reconstructed using the attributes outlook and humidity.[1] IDL finds that humidity and outlook both have a topological relevance gain of .5. Outlook is the highest up in the tree and is pulled up to the root (step 3.5). The subtree on the branch outlook=overcast is pruned (step 3.7):

[1] It is important to compute the set of attributes A_i at each level of the tree after revision of the lower parts of the path. For example, before the previous transformation humidity was not one of them, but now it is.



The next example is example 1. Note that it is now classified correctly: transformation did increase accuracy. The classification path goes in the highly symmetrical subtree with windy at the root. IDL detects the topological relevance of humidity at that node for example 1 and pulls it up (step 3.5). After this transformation, which was illustrated in section 2, pruning is possible (step 3.7). IDL still has to revise the test at the root but finds no better attribute. The tree is topologically minimal so that none of the examples leads to further revisions. It should be stressed that, though in this case IDL finds the same tree (the best one) as the other algorithms, it does so in a very different manner.

4. Analysis and Empirical results

4.1. Complexity and Convergence

In [12] I analyze worst-case order of magnitudes for training cost. As in [11] this is decomposed as the number of instance-count additions (ica) and the number of evaluations of the statistical criterion (E-score), and expressed in the number of attributes |A| and the maximal branching factor b. Table 4 summarizes the results of the analysis of training cost per example for ID5, ID5R and IDL.

Table 4: Worst Case Training Cost per Example

	Icas	E-scores
ID5	$O(b^{\|A\|})$	$O(\|A\|^2)$
ID5R	$O(\|A\|.b^{\|A\|})$	$O(b^{\|A\|})$
IDL	$O(\|A\|.b^{\|A\|})$	$O(\|A\|^2)$

As with ID5 and ID5R one pass over the examples is sufficient to build a correct tree with IDL. However what is important for IDL is the number of examples required to converge to the optimal tree when examples are randomly seen. Optimal means correct, topologically minimal, cannot be pruned without loss of accuracy and does not contain single-branch splits (see section 3). Analysis is lacking here but experiments have shown that for some concepts and certain training histories, IDL may fail to find the optimal tree even if it exists. Elomaa [1] shows and explains cases of non-convergence on the 3-multiplexer. No case has been observed where IDL never finds the optimal tree if it exists, so multiple runs of the algorithm may be a solution. It is easily seen from the algorithm in table 3 that, if IDL ever generates a correct tree which is topologically minimal then it will stop performing transformations. This and the experimental results support the conjecture (as yet not proven) that, if a topologically optimal tree exists then with non-zero probability IDL finds it and in contrast to ID5(R) sticks to it. If there exist multiple such trees then IDL finds one of them, though not necessarily the smallest in terms of number of nodes (use multiple runs). If no topologically minimal tree exists then IDL will not converge to a unique tree. Typically, with each example the tree is reconfigured to lower the topological relevance of an attribute, thereby increasing it for another one. Some and possibly all trees in this limit cycle have lowest possible topological relevance for the set of examples, though never all one or zero (which would mean convergence to a unique tree).

4.2. Experimental Results

I did experiments to compare IDL and ID5R[2]. The results are stated in terms of the evolution of tree complexity (number of nodes and number of leaves) and accuracy with increasing number of training examples. Training is done in groups of examples randomly chosen with replacement. Measurements are averaged over 20 runs. The characteristic behavior of IDL is to grow a tree which is far too large but then rapidly collapses.

Figure 1 shows the results for the weather concept from table 1. There is a unique topologically minimal tree (8 nodes, 5 leaves). IDL has 20 correct and topologically minimal trees after 70 examples. ID5R is equally fast in accuracy, but does not always find the minimal tree (average of 8.7 nodes and 5.4 leaves). In numerous runs on this concept IDL never failed to find the topologically minimal tree.

Figure 2 shows the results for the 6-multiplexer. There are two topologically minimal trees which are essentially equivalent (15 nodes and 8 leaves). For IDL all 20 trees

[2] ID5R is used with postpruning to remove unnecessary splits after transformation (see [9] p111). The measurements for ID5 are hardly different from those for ID5R.

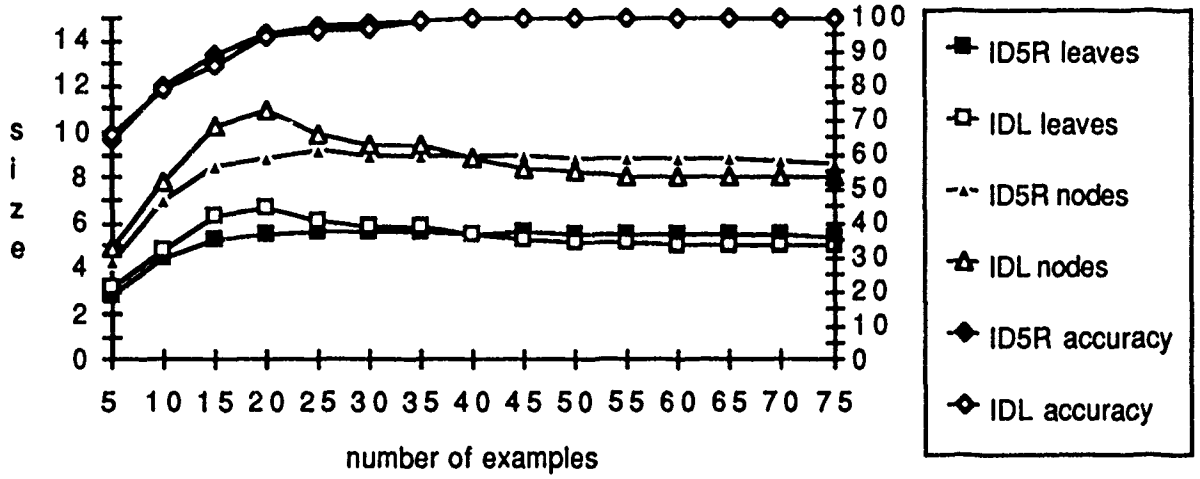


Figure 1: IDL (white marks) and ID5R (black marks) averaged over 20 runs on the weather concept (see table 1). Training per 5. Left axis: tree size plotted in squares (number of leaves) and triangles (number of nodes). Right axis: accuracy with respect to all examples, plotted with circles.

are fully accurate after 110 examples, and topologically minimal after 150 examples. After 200 examples ID5R has an average of 41.6 nodes, 21.3 leaves and 99.7 accuracy. In numerous runs on this concept IDL never failed to find a topologically minimal tree. IDL has also been run on the 11-multiplexer which uses 3 address-bits and 8 data-bits. There are 2048 examples. The optimal tree has 16 leaves and 31 nodes. IDL had a correct tree after (less than) 650 examples and the minimal one after (less than) 700 examples. Surprisingly IDL sometimes fails to find the minimal tree for the 3-multiplexer (about 40% hit-rate). Often it limit-cycles between the two smallest TDIDT equivalent trees (see also [1]).

The concept of 3-parity has 6 smallest trees with the same topology, but no topologically minimal one (TR-scores 1,1,2 for all examples). The graphs are similar to

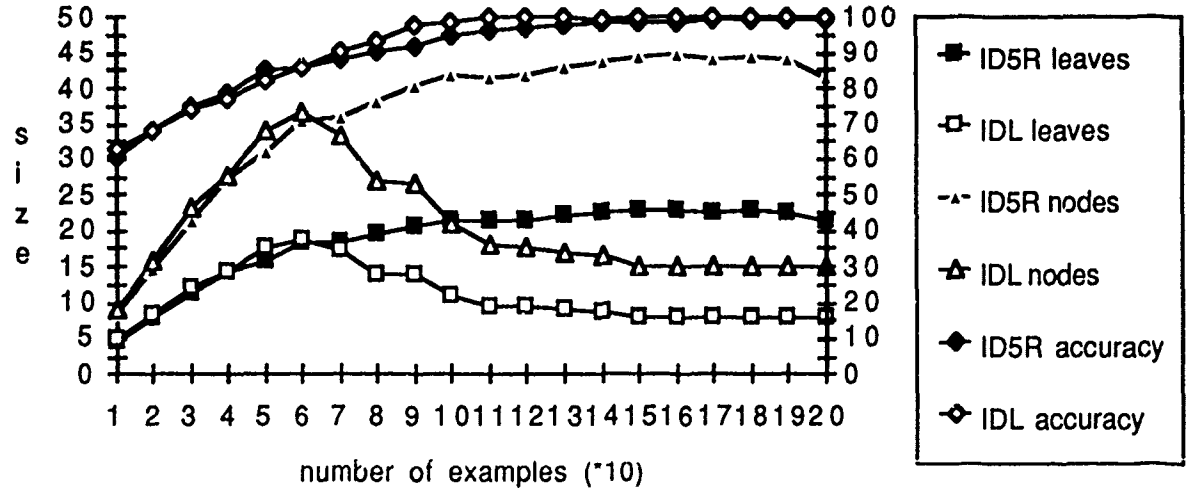


Figure 2: IDL (white marks) and ID5R (black marks) averaged over 20 runs on the 6-multiplexer. Training per 10. Left axis: tree size plotted in squares (number of leaves) and triangles (number of nodes). Right axis: accuracy with respect to all examples, plotted with circles.

those in figure 1. However, IDL goes into a limit cycle of three smallest trees (15 nodes, 8 leaves) with every example permuting attributes through the tree levels. ID5R goes to a slower cycle of all 6 trees.

I used IDL as a postprocessor for optimizing TDIDT generated trees while in use for classification. The results for the 6-multiplexer are shown in figure 3. They show how IDL rapidly collapses the trees to a topologically minimal form. Much the same effect can be achieved without using any statistical information or examples for focus, but purely by analyzing the occurrences of the attributes on a path from root to leave. Experiment like this with a variant of IDL are reported in [1].

Table 5 shows the run-time improvement of IDL over ID5R expressed in icas, E-scores, expansions, prunings and attribute switches. The figures express reduction in total training cost, averaged over 20 consecutive examples, and averaged over 20 runs on weather concept (examples 55-75) and 6-multiplexer (examples 130-150). Note that IDL always found the same or a better tree.

Table 5: Computation Reduction with Respect to ID5R

	weather	6-multiplexer
Icas	11%	29%
E-scores	6%	32%
Expansions	40%	66%
Prunings	44%	66%
Transformations	37%	49%

4.3. Discussion

Despite the fact that IDL sometimes fails to converge to a topologically minimal tree it performs well on a number of standard concepts. There may be two reasons for this. Firstly, the topological flavor of IDL makes it less sensitive to statistical variations (non-representative example distributions). This is well illustrated by the weather concept where ID5R sometimes converges to a tree with humidity as root. For the same training history, IDL is not distracted by these variations. Secondly IDL uses individual examples to guide its search but actually reasons about classification paths which represent classes of examples. This may explain the fast convergence on the 6-multiplexer for which these classes are fairly large. Note that IDL's reliance on individual examples likely makes it very sensitive to noise.

5. Related and Future Work

5.1. Related Work

IDL is a direct descendant of earlier incremental decision tree algorithms ID4 [7], ID5 [9] and ID5R [11]. In section 3 it was noted that these do not tackle the problem of suboptimal trees, because they try to stick to the search path which TDIDT would follow and thus inherit from TDIDT the problem of suboptimality. Selection measures for TDIDT algorithms have been improved to generate smaller trees without loss of accuracy [2,5]. However, Quinlan [6] shows why TDIDT algorithms have problems with concepts like the multiplexer. Seshu [8] shows that TDIDT algorithms are fundamentally incapable of effectively learning a class of generalized parity concepts. Remedies fall in two categories: subsequent simplification of trees, or change of language bias by introducing new attributes.

Pruning techniques [3] allow one to change a tree near the fringe. For concepts like the multiplexer this is not sufficient. Quinlan [6] proposes to transform a tree into a set of rules which are subsequently simplified by selectively deleting conditions. For comparison, note that IDL works with one representation and that the pull-up process modifies several rules at once and is capable of

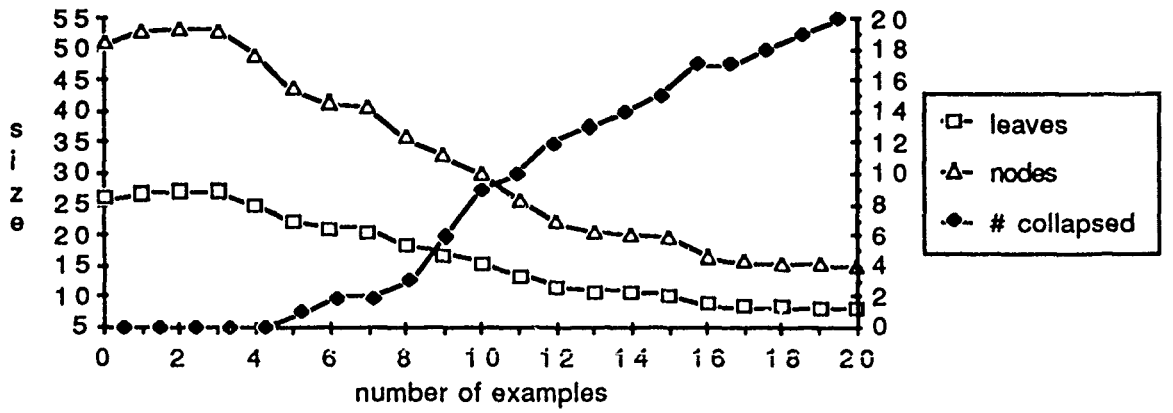


Figure 3: IDL as a postprocessor for TDIDT-generated trees, applied to 20 identical trees for the 6-multiplexer. Left axis: tree size plotted in squares (number of leaves) and triangles (number of nodes). Right axis: the number of fully collapsed trees at that moment, plotted with black circles.

both introducing and deleting attributes on a path. Pruning and Quinlan's technique solve problems of noise, while IDL's reliance on individual instances makes it sensitive to noise.

The FRINGE algorithm [4] is designed to tackle the problem of replication of subtrees when learning decision trees for boolean Disjunctive Normal Form concepts by introducing macro-attributes. Like IDL this algorithm has a topological flavor and uses decision trees in a bottom up way. It is however not incremental and takes TDIDT as its gold-standard. In comparison note that IDL is incremental, does not change language bias and tackles the replication problem for concepts which do have a representation without replication. Seshu [8] also introduces macro-attributes, though not based on an analysis of tree topology.

Utgoff [10] exploits the incremental nature of ID5R to increase the quality of learning. The idea is to learn only from those examples which are wrongly classified. Tree size improves though for the 6-multiplexer is still far from optimal (31 nodes). This idea does not seem applicable to IDL which does most simplification work after the tree has reached full accuracy.

Elomaa [1] presents experimental results to use a variant of IDL (without step 3.6) as a postprocessor to optimize TDIDT generated decision trees. Their results show that for exclusive or functions IDL efficiently removes tests for irrelevant attributes.

5.2. Future Work

The role of the statistical measure in step 3.6 of the algorithm needs to be better understood, in particular with respect to the goodness which is required of it. I used IDL with random attribute selection, fixed order selection and second best according to information gain. In all these cases IDL did not converge. I did not check whether IDL improves with some of the selection measures from [2]. Further analysis and experimentation are needed to derive average case estimates for computation time and complexity and to compare these with the worst case estimates (finding optimal trees is in general NP-complete...). Run-time measurements must be analysed in greater detail. The IDL algorithm needs to be extended to detect and handle non-convergence. Finally issues of noise have not been investigated yet.

6. Conclusion

This paper has introduced the notion of topological relevance as a measure for the complexity of a decision tree representation. It has presented an algorithm called IDL for incremental induction of topologically minimal trees. The key idea is to use tree transformations to create opportunities for pruning, guided by an analysis of tree topology. Empirical results show that often IDL rapidly finds a minimal tree if it exists. This work is continued with experiments, analysis and improvements to cope with non-convergence and noise.

Acknowledgements

Thanks to Johan Vanwelkenhuysen, Philip Rademakers, Jeff Schlimmer, Paul Utgoff, Giulia Pagallo and the anonymous reviewers for suggesting various improvements. This work was supported by IWONL contract No. 4465 and currently by "Dirección General de Investigación Científica y Técnica" (DGICYT), Spain.

References

[1] Elomaa, T., Kivinen, J. (1990) On Inducing Topologically Minimal Decision Trees. *(Unpublished, Submitted to AAAI-90).*

[2] Mingers, J. (1989) An empirical comparison of selection measures for decision-tree induction. In *Machine Learning* 3, p319-342. Kluwer Academic Publishers. Boston, MA.

[3] Mingers (1989) An empirical comparison of pruning methods for decision-tree induction. In *Machine Learning* 4, p227-243. Kluwer Academic Publishers. Boston, MA.

[4] Pagallo, G., Haussler, D. (1989) Two Algorithms that Learn DNF by Discovering Relevant Features. In Segre, A.M. (Ed.) *Proceedings of the Sixth International Workshop on Machine Learning* p119-123. Morgan Kaufmann. San Mateo, CA.

[5] Quinlan, J.R. (1986) Induction of Decision Trees . In *Machine Learning* 1, p81-106. Kluwer Academic Publishers. Boston, MA.

[6] Quinlan, J.R. (1988) An empirical comparison of genetic and decision-tree classifiers. In Laird, J. (Ed.) *Proceedings of the Fifth International Conference on Machine Learning* p135-141. Morgan Kaufmann. San Mateo, CA.

[7] Schlimmer, J.C., Fisher, D. (1986) A case study of incremental concept induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence* p496-501. Morgan Kaufmann. San Mateo, CA.

[8] Seshu, R. (1989) Solving the Parity Problem. In Morik, K. (Ed.) *Proceedings of the fourth European Working Session on Learning*, p263-271. Pitman, London.

[9] Utgoff, P.E. (1988) ID5: An Incremental ID3. In Laird, J. (Ed.) *Proceedings of the Fifth International Conference on Machine Learning* p107-120. Morgan Kaufmann. San Mateo, CA.

[10] Utgoff, P.E. (1989) Improved Training via Incremental Learning. In Segre, A.M. (Ed.) *Proceedings of the Sixth International Workshop on Machine Learning* p362-365. Morgan Kaufmann. San Mateo, CA.

[11] Utgoff, P.E. (1989) Incremental Learning of Decision Trees. In *Machine Learning* 4, p161-186. Kluwer Academic Publishers. Boston, MA.

[12] Van de Velde, W. (1990) Incremental Learning of Optimal Decision Trees. Unpublished manuscript.

CONCEPTUAL CLUSTERING

A Rational Analysis of Categorization

John R. Anderson
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

Michael Matessa
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

A rational analysis tries to predict the behavior of a cognitive system from the assumption it is optimized to the environment. An iterative categorization algorithm has been developed which attempts to get optimal Bayesian estimates of the probabilities that objects will display various features. A prior probability is estimated that an object comes from a category and combined with conditional probabilities of displaying features if the object comes from the category. Separate Bayesian treatments are offered for the cases of discrete and continuous dimensions. The resulting algorithm is efficient, works well in the case of large data bases, and replicates the full range of empirical literature in human categorization.

A rational analysis (Anderson, 1990) is an attempt to specify a theory of some cognitive domain by specifying the goal of the domain, the statistical structure of the environment in which that goal is being achieved, and whatever computational constraints the system is operating under. The predictions about the behavior of the system can be derived assuming that the system will maximize the goals it expects to achieve while minimizing expected costs where expectation is defined with respect to the statistical structure of the environment. This approach is different from most approaches in cognitive psychology because it tries to derive a theory from assumptions about the structure of the environment rather than assumptions about the structure of the mind.

We have applied this approach to human categorization and have developed a rather effective algorithm for categorization. The analysis assumes that the goal of categorization is to maximize the accuracy of predictions about features of new objects. For instance, one might want to predict whether an object is dangerous or not. This approach to categorization sees nothing special about category labels. The fact an object might be called a tiger is just another feature one might want to predict about the object.

The Structure of the Environment

It is an interesting question what kind of structure we can assume of the environment in order to drive prediction. The theory developed rested on the structure of biological categories produced by the phenomenon of species. Species form a nearly disjoint partitioning of the natural objects because of the inability to interbreed. Within a species there is a common genetic pool which means that individual members of the species will display particular feature values with probabilities that reflect the proportion of that phenotype in the population. Another useful feature of species structure is that the display of features within a freely-interbreeding species is largely independent. Thus, there is little relationship between size and eye color in species where those two dimensions vary. Thus, the critical aspects of speciation is the disjoint partitioning of the object set and the independent probabilistic display of features within a species.

An interesting question is whether other types of objects display these same properties. Another common type of object is the artifact. Artifacts approximate a disjoint partitioning but there are occasional exceptions--for instance, mobile homes which are both homes and vehicles. Other types of objects (stones, geological formations, heavenly bodies, etc) seem to approximate a disjoint partitioning but here it is hard to know whether this is just a matter of our perceptions or whether there is any objective sense in which they do. One can use the understanding of speciation for natural kinds and understanding of the intended function in manufacture for artifacts to objectively assess the hypothesis of a disjoint partitioning.

We have taken this disjoint, probabilistic model of categories and used it as the understanding of the structure of the environment for doing prediction about object features. To maximize the prediction of features of objects we need to induce a disjoint partitioning of the object set into categories and determine what the probability of features will be for each category. The ideal prediction function would be described by the following formula:

$$Pred_{ij} = \sum_{x} P(x|F_n) Prob_i(j|x)$$

where $Pred_{ij}$ is the probability an object will display a value j on a dimension i which is not observed for that object, the summation is across all possible partitionings of the n objects seen into disjoint sets, $P(x|F_n)$ is the probability of partitioning x given the objects display observed feature structure F_n, and $Prob_i$ (j|x) is the probability the object in question would display value j on dimension i if x were the partition. The problem with this approach is that the number of partitions of n objects grows exponentially as the Bell exponential number (Berge, 1971). Assuming that humans cannot consider an exponentially exploding number of hypothesis we were motivated to explore iterative algorithms such as those developed by Fisher (1987) and Lebowitz (1987).

The following is a formal specification of the iterative algorithm:

1. Before seeing any objects, the category partitioning of the objects is initialized to be the empty set of no categories.
2. Given a partitioning for the first *m* objects, calculate for each category *k* the probability P_k that the *m+1st* object comes from category *k*. Let P_o be the probability that the object comes from a completely new category.
3. Create a partitioning of the *m+1* objects with the *m+1st* object assigned to the category with maximum probability.
4. To estimate the probability of value *j* on dimension *i* for the *n+1st* object calculate

$$Pred_{ij} = \sum_k P_k \; P(ij|k) \qquad \text{Equation 1}$$

where P_k is the probability the *n+1st* object comes from category *k* and *P(ij/k)* is the probability of displaying value *j* on dimension *i*.

The basic algorithm is one in which the category structure is grown by assigning each incoming object to the category it is most likely to come from. Thus, a specific partitioning of the objects is produced. Note, however, that the prediction for the new *n+1st* object is not calculated by determining its most likely category and the probability of *j* given that category. This calculation is performed over all categories. This gives a much more accurate approximation to the ideal $Pred_{ij}$ because it handles situations where the new object is ambiguous between multiple categories. It will weight approximately equally these competing categories.

The algorithm is not guaranteed to produce the maximally probable partitioning of the object set since it only considers partitionings that can be incrementally grown. It also does not weight multiple possible partitionings as the ideal algorithm would. In cases of strong category structure, there will be only one probable partitioning and the iterative algorithm will uncover it. In cases of weak category structure, it will often fail to obtain the ideal partitioning, but still the predictions obtained by Equation 1 closely approximate the ideal quantity because of the weighting of multiple categories. We observe correlations about .95 between the predictions of our algorithm and the ideal quantities in cases of small data sets.

It remains to come up with a formula for calculating P_k and *P(ij/k)*. Since *P(ij/k)* proves to be involved in the definition of P_k, we will focus on P_k. In Bayesian terminology P_k is a posterior probability *P(k/F)* that the object belongs to category *k* given that it has feature structure *F*. Bayes formula can be used to express this in terms of a prior probability *P(k)* of coming from category *k* before the feature structure is inspected and a conditional probability *P(F/k)* of displaying the feature structure *F* given that it comes from category *k*.

$$P_k = P(k|F) = \frac{P(k)P(F|k)}{\sum_k P(k)P(F|k)} \qquad \text{Equation 2}$$

where the summation in the denominator is over all categories k currently in the partitioning including the potential new one. This then focuses our analysis on the derivation of a prior probability *P(k)* and a conditional probability *P(F/k)*.

Prior Probability

With respect to prior probabilities the critical assumption is that there is a fixed probability *c* that any two objects come from the same category and this probability does not depend on the number of objects seen so far. This is called the coupling probability. If one takes this assumption about the coupling probability between two objects being independent of the other objects and generalizes it, one can derive a simple form for *P(k)* (See Anderson, 1990, for the derivation):

$$P(k) = \frac{cn_k}{(1-c) + cn} \qquad \text{Equation 3}$$

where *c* is the coupling probability, n_k is the number of objects assigned to category *k* so far, and *n* is the total number of objects seen so far. Note for large *n* this closely approximates n_k/n which means that we have a strong base rate effect in these calculations with a bias to put new objects into large categories. Presumably the rational basis for this is apparent.

We also need a formula for *P(0)* which is the probability that the new object comes from an

entirely new category. This is

$$P(O) = \frac{(1-c)}{(1-c) + cn} \qquad \text{Equation 4}$$

For large n this closely approximates $(1\text{-}c)/cn$ which is again a reasonable form--i.e., the probability of a brand new category depends on the coupling probability and number of objects seen. The greater the coupling probability and the more objects, the less likely it is that the new object comes from an entirely new category.

Conditional Probability

We can consider the probability of displaying features on various dimensions given category membership to be independent of the probabilities on other dimensions. Then we can write

$$P(F|k) = \prod_i P(ij|k) \qquad \text{Equation 5}$$

Where P(ij|k) is the probability of displaying value j on dimension i given that one comes from category k.

This independence assumption does not prevent us from recognizing categories with correlated features. Thus, we may know that being black and retrieving sticks are features found together in labradors. This would be represented by high probabilities of the stick-retrieving and the black features in the labrador category. What the independence assumption prevents us from doing is representing categories where values on two dimensions are either both one way or both the opposite. Thus, it would prevent us from recognizing a single category of animals which were either large and fierce or small and gentle, for instance. However, this turns out not to be a very serious limitation. What our algorithm does in this case is to spawn a different category to capture each two-feature combination--it would create a category of large and fierce creatures and another category of small and gentle creatures.

The effect of Equation (5) is to focus us down on an analysis of the individual $P(ij|k)$. Derivation of this quantity is itself an exercise in Bayesian analysis. We will treat separately discrete and continuous dimensions.

Discrete Dimensions

The basic Bayesian strategy for doing inference along a dimension is to assume a prior distribution of values along the dimension, determine the conditional probability of the data under various possible values of the priors, and then calculate a posterior distribution of possible values. The common practice is to start with a rather weak distribution of possible priors and as more and more data accumulates come up with a tighter and tighter posterior distribution.

In the case of a discrete dimension, the typical Bayesian analysis (Berger, 1985) is to assume that the prior distribution is a Dirichlet density. For a dimension with m values a Dirichlet distribution is characterized by m parameters α_j. We can define $\alpha_o = \sum_j \alpha_j$. The mean probability of the jth value is $p_j = \alpha_j/\alpha_o$. The value α_o reflects the strength of belief in these priors probabilities, p_j. The data after n observations will consist of a set of C_j counts of observations of value j on dimension i. The posterior distribution of probabilities is also a Dirichlet distribution but with parameters α_j+C_j. This implies that the mean expected value of displaying value j in dimension i is $(\alpha_j+C_j)/\sum(\alpha_j+C_j)$. This is P(ij|k) for Equation 5:

$$P(ij|k) = \frac{C_j + \alpha_i}{n_k + \alpha_0} \qquad \text{Equation 6}$$

where n_k is the number of objects in category k which have a value on dimension i and C_j is the number of objects in category k with the same value as the object to be classified. For large n_k this approximates C_j/n_k which one frequently sees promoted as the rational probability. However, it has to have this more complicated form to deal with problems of small samples. For instance, if one has just seen one object in a category and it has had the color red, one would not want to guess that all objects are red. If we assume there are seven colors and all the α_j were 1, the above formula would give 1/4 as the posterior probability of red and 1/8 for the other six colors unseen as yet.

Continuous Dimensions

Application of Bayesian inference schemes to continuous dimensions is more problematic but there is one approach that appears most tractable (Lee, 1989). The natural assumption is that the variable is distributed normally and the induction problem is to infer the mean and variance of that distribution. In standard Bayesian inference methodology we must begin with some prior assumptions about what the mean and variance of this distribution is. It is unreasonable to suppose we can know in advance what the precisely what either the mean and variance will be. Our prior knowledge must take the form of probability densities over possible means and variances. This is basically the same idea as in the discrete case where we had a Dirichlet distribution giving priors about probabilities of various values. The major complication is the need to state separately prior distributions for mean and variance.

The tractable suggestion for the prior distributions is that the inverse of the variance Σ^2 is distributed according to a chi-square distribution and the mean has a normal distribution. Given these priors, the posterior distribution of values x on a continuous dimension *i* for category k, after n observations has the following t distribution:

$$f_i(x|k) \sim t_{a_i}(\mu_i, \sigma_i \sqrt{1 + 1/\lambda_i}) \qquad \text{Equation 7}$$

The parameters a_i, μ_i, σ_i, and λ_i are defined as follows:

$$\lambda_i = \lambda_0 + n \qquad \text{Equation 8}$$

$$a_i = a_0 + n \qquad \text{Equation 9}$$

$$\mu_i = \frac{\lambda_0 \mu_0 + n\bar{x}}{\lambda_0 + n} \qquad \text{Equation 10}$$

$$\sigma_1^2 = \frac{a_0\sigma_0 + (n-1)s^2 + \frac{\lambda_0 n}{\lambda_0 + n}(\bar{x} - \mu_0)^2}{a_0 + n} \qquad \text{Equation 11}$$

where $\bar{x}$ is the mean of the n observations and s^2 is their variance. These equations basically provide us with a formula for merging the prior mean and variance, μ_0 and σ_0^2, with the empirical mean and variance, $\bar{x}$ and s^2, in a manner that is weighted by our confidences in these priors, λ_0 and a_0.

Equation 7 for the continuous case describes a probability density which serves the same role as Equation 6 for the discrete case which describes a probability. The product of conditional probabilities in Equation 5 will then be a product of probabilities and density values. Basically, Equations (5), (6), and (7) give us a basis for judging how similar an object is to the category's central tendency.

Conclusion

This completes our specification of the theory of categorization. Before looking at its application to various empirical phenomena a word of caution is in order. The claim is not that the human mind performs any of the Bayesian mathematics that fills the preceding pages. Rather the claim of the rational analysis is that, whatever the mind does, its output must be optimal. The mathematical analyses of the preceding pages serve the function of allowing us, as theorists, to determine what is optimal.

A second comment is in order concerning the output of the rational analysis. It delivers a probability that an object will display a particular feature. There remains the issue of how this relates to behavior. Our basic assumption will only be that there is a monotonic relationship between these probabilities and behavioral measures such as response probability, response latency, and confidence of response. The exact mapping will depend on such things as the subject's utilities for various possible outcomes, the degree to which individual subjects share the same priors and experiences, and the computational costs of achieving various possible mappings from rational probability to behavior. These are all issues for future exploration. What is remarkable is how well we can fit the data simply assuming a monotonic relationship.

Application of the Algorithm

We have applied the algorithm to a number of examples to illustrate its properties. The algorithm is quite efficient. A Franz LISP implementation categorized the 290 items from Michalski and Chilausky's data set on Soybean disease (each with 36 values) in 1 CPU minute on a Vax 780 or a MAC II. This is without any special effort to optimize the code. It also diagnosed the test set of 340 soybean instances with as much accuracy as apparently did the original system of Michalski and Chilausky.

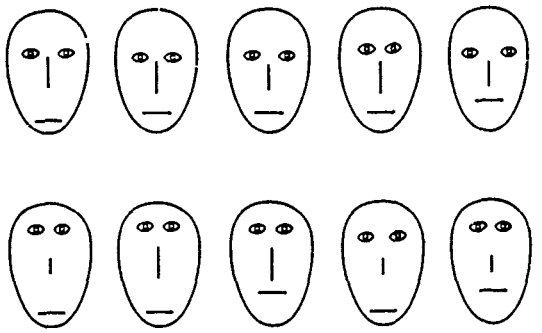
The algorithm has been applied to the full range of psychological experiments in categorization. Detailed discussions can be found in Anderson (in press) and Anderson & Matessa (in preparation). However, we will review here in varying detail the applications of the algorithm to 10 empirical phenomena. All these simulations were done with a constant setting of the parameters: c from Equation 3 and 4 at .3, α_i from Equation 6 at 1, λ_0 from Equation 8 at 1, a_0 from Equation 9 at 1, μ_0 from Equation 10 at the mean of the stimuli, and σ_0^2 from Equation 11 at the square of 1/4 the stimulus range. All of these are plausible settings and often correspond to conventions for setting Bayesian non-informative priors. The following are among the empirical phenomena we have successfully simulated:

1. Extraction of Central Tendencies, Continuous Dimensions The Bayesian model for continuous dimensions implies that categorization should vary with distance from central tendency. This enables the model to simulate the data of Posner & Keele (1968) on categorization of dot patterns and Reed (1972) on categorization of faces. Let us consider the experiment of Reed in a little detail:

Reed (1972) had subjects learn to categorize the 10

faces which are illustrated in Figure 1. The first row of faces are in one category and the second row of faces are in another category. The two sets of faces are deviations from underlying prototypes. After studying these faces subjects went to a test condition where they had to try to classify these and other faces. The critical data concerns the probability with which subjects assigned faces to conditions. As a general characterization, their categorization varied with distance of the face from the prototype.

Figure 1

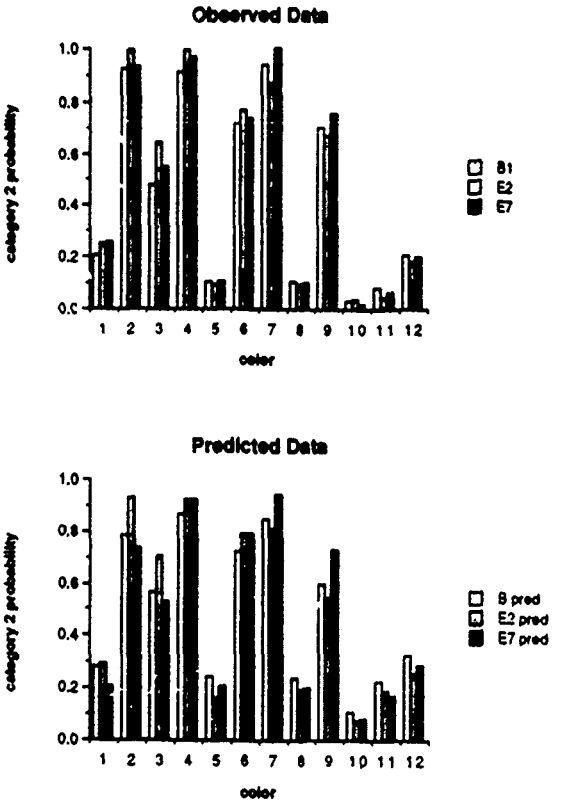


In our attempt to simulate these data we treated these faces as five-dimensional stimuli where the dimensions are height of the forehead which ranged from 54 to 88 mm, distance separation of the eyes which ranged from 20 to 55 mm, length of the nose which ranged from 32 to 64 mm, height of the mouth which ranged from 28 to 60 mm, and category label which was a binary-valued discrete dimension. Our rational model identified two or more internal categories, depending on presentation order, that corresponded to the experimenter's categories. That is, sometimes it subdivided the experimenter's categories into subcategories but it almost never merged items from the two experimenter categories into an internal category. Reed's subjects were asked to classify 25 test stimuli and the major test of our model was its classification of these test stimuli. Overall its confidence of category membership (calculated by Equation 1) correlated .90 with Reed's data.[1]

2. Extraction of Central Tendencies, Discrete Dimensions The model implies that stimuli should be better categorized if they display the majority value for a dimension. This enabled the model to simulate the data of Hayes-Roth & Hayes-Roth (1977), for instance.

3. Effect of Individual Instances If an instance is sufficiently different than the central tendency for its assigned category, the model will form a distinct category for it. This enables the model to account for the data of Medin & Schaffer (1978) on discrete dimensions and Nosofsky (1988) on continuous dimensions. Let us consider the experiment of Nosofsky:

Figure 2



Nosofsky trained his subjects on 12 stimuli that varied in brightness and saturation. The colors varied in brightness on the Munsell scale from 3 to 7 and in saturation from 4 to 12. In the base condition subjects had four trials on each item and were then tested. In the first experiment there was a condition E2 in which subjects saw stimulus 2 approximately 5 times as frequently and a condition E7 in which they saw stimulus 7 approximately 5 times as frequently. Part (a) of Figure 2 illustrated probability of classification in Category 2. As can be seen subjects are sensitive to the frequency manipulation. Part (b) of Figure 2 shows the probability our model assigned to a Category 2 response given the same experience. The overall correlation between data and theory is .98.

4. Linearly Separable versus Non-Linearly Separable Categories Unlike some categorization models this model is able to learn categories that cannot be separated by a plane in a n-dimensional

[1]We would like to thank Stephen Reed for making his data available.

hyperspace. This is because it can form multiple internal categories to correspond to an experimenter's category. This enables the model to account for the data of Medin & Schwanenflugel (1981) on discrete dimensions and Nosofsky, Clark, & Shin (1989) on continuous dimensions. Let us consider the experiment of Medin & Schwanenflugel. They performed an experiment where linearly non-separable categories were learned better than linearly separable categories.

Table 1 illustrates the material used by Medin & Schwanenflugel (1981). In the case of the linearly separable categories our model formed separate categories for each stimulus. In the case of linearly non-separable, it merged the first 2 in category A into an internal category, the second 2 in category A, and the first, second, and fourth in category B. Thus, only stimulus 3 in category B was in a singleton category and this was the stimulus that produced the highest error rate in the non-separable condition.

Table 1

LINEARLY SEPARABLE CATEGORIES

CATEGORY A					CATEGORY B				
	DIMENSION					DIMENSION			
EXEMPLAR	D_1	D_2	D_3	D_4	EXEMPLAR	D_1	D_2	D_3	D_4
A_1	1	1	1	0	B_1	1	0	1	0
A_2	1	0	1	1	B_2	0	1	1	0
A_3	1	1	0	1	B_3	0	0	0	1
A_4	0	1	1	1	B_4	1	1	0	0

CATEGORIES NOT LINEARLY SEPARABLE

CATEGORY A					CATEGORY B				
	DIMENSION					DIMENSION			
EXEMPLAR	D_1	D_2	D_3	D_4	EXEMPLAR	D_1	D_2	D_3	D_4
A_1	1	0	0	0	B_1	0	0	0	1
A_2	1	0	1	0	B_2	0	1	0	0
A_3	1	1	1	1	B_3	1	0	1	1
A_4	0	1	1	1	B_4	0	0	0	0

5. Basic-Level Categories The internal categories that the model extracts corresponds to what Rosch (1976) meant by basic-level categories.[2] Thus, it can simulate the data of Murphy & Smith (1982) and Hoffman & Ziessler (1983). We will describe the application to Murphy and Smith.

Murphy and Smith presented to their subjects 16 objects identified as examples of fictitious tools. The structure of the material, as encoded by Gluck and Corter (1985), is illustrated in Table 2. There were two superordinate categories which divided into 4 intermediate categories, which divided into 8 subordinate categories. Table 2 gives the attribute description of each category. Subjects were fastest to classify the material at the intermediate level which Murphy and Smith intended to be the basic level. Objects at this level had two attributes plus two labels in common. Only one additional feature and label was gained at the subordinate level, and all features were lost at the superordinate level except for their feature of being a pounder or a cutter.

Table 2

Gluck and Corter's Analysis of the Feature Structure of the Material from Murphy & Smith (1982)

	Categories			*Attributes*			
Item #	*Super-ordinate*	*Inter-mediate*	*Sub-ordinate*	*Handle*	*Shaft*	*Head*	*Size*
1.	Pounder	Hammer	Hammer1	2	2	0	0
2.				2	2	0	1
3.			Hammer 2	2	2	1	0
4.				2	2	1	0
5		Brick	Brick1	0	3	4	0
6				0	3	4	1
7.			Brick 2	1	3	4	0
8.				1	3	4	1
9.	Cutter	Knife	Knife1	3	4	2	0
10.				3	4	2	1
11.			Knife2	3	4	3	0
12.				3	4	3	1
13.		Pizza C.	P.C.1	4	0	5	0
14.				4	0	5	1
15.			P.C.2	4	1	5	0
16.				4	1	5	1

We modeled this material by encoding the stimuli as 7-dimensional objects with dimensions for the superordinate label (2 values), the intermediate label (4 values), the subordinate label (8 values), handle (5 values), shaft (5 values), head (6 values), and size (2 values). What category structure was obtained depended upon the value of the coupling probability. For $c > .96$ all were merged into one category; for $.95 > c > .8$ the two superordinate categories emerged; for $.8 > c > .4$ the model fluctuated between the superordinate and intermediate categories depending on presentation order; for $.4 > c > .2$ it extracted just the intermediate categories; for $.2 > c > .05$ it basically extracted the intermediate categories with an occasional singleton category or subordinate category; for $c < .05$ it extracted only singleton categories. In summary, the subordinate categories never emerged and only a very high levels of c did superordinate categories dominate. At the value of c used in the simulations of this paper ($c = .3$) only the basic level categories emerged. Thus, it seems fair to conclude that the analysis agrees with the subjects as to what the basic level is.

6. Probability Matching Faced with truly probabilistic categories and large samples of instances the model will estimate probability of

[2]Rosch's idea of a basic level is that there is a level in the generalization hierarchy to which we first assign objects. For instance, she argues we would first see an object as a bird not a sparrow or an animal.

features that correspond exactly to the empirical proportion. Thus, it predicts the data of Gluck & Bower (1988) on probability matching.

7. Base-Rate Effect Because of Equation 3 this model predicts that usually there will be a greater tendency to assign items to categories of large size. Thus, it handles the data of Homa & Cultice (1984). It also reproduces the more subtle interactions of Medin & Edelson (1988).

8. Correlated Features As noted earlier the model can handle categories with correlated features by breaking out separate internal categories for each feature combination. Thus, it handles the data of Medin, Altom, Edelson, & Freko (1982). They had subjects study the 9 cases in Table 3 which were all supposed to represent instances from one disease category, burlosis. This was simulated by presenting these 9 cases to the model with a sixth dimension, a disease label which was always burlosis. This was arbitrarily treated this as a binary dimension. Note that each of the five symptoms show a majority of ones associated with the disease.

Table 3

SYMPTOMS OF BURLOSIS
from Medin et al. (1982)

Case Study	Blood Pressure	Skin Condition	Muscle Condition	Condition of Eyes	Weight Condition
1. R.L.	0	1	0	1	1
2. L.F.	1	1	0	1	1
3. J.J.	0	0	1	1	1
4. R.M.	1	0	1	1	1
5. A.M.	1	1	1	1	1
6. J.S.	1	1	1	1	1
7. S.T.	1	0	0	0	0
8. S.E.	0	1	1	0	0
9. E.M.	1	1	1	0	0

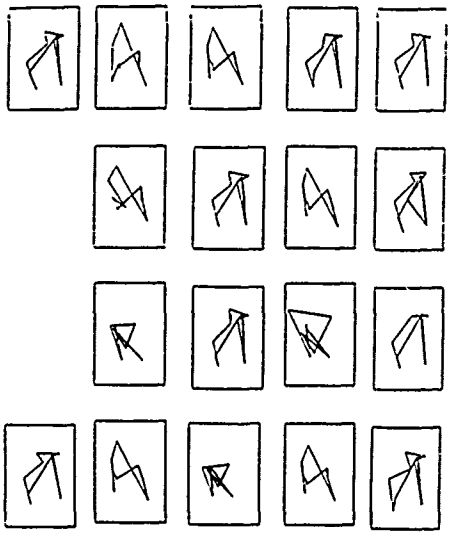
Note Zero denotes absence of the symptom and 1 denotes presence

The critical feature of these materials from the perspective of correlated features concerns the fourth dimension of conditions of eyes and the fifth dimension of weight. Values are either both 1 or both 0. The first six items in Table 3.5 have two 1's; the last three have two 0's. Subjects are sensitive to this correlation. When these stimuli were fed into the algorithm with $c=.3$, it typically extracted 3 categories--one to represent the first six items, one for the seventh, and one for the last two. Thus, the way it dealt with correlated features was to break out separate categories for the different possible values of the correlation.

9. Effects of Feedback If the category structure of the stimuli is strong enough the model can extract the categories without any feedback as to category identity. In the face of weak category structure, it is necessary to provide category labels to get learning. Thus, this model reproduces the data of Homa & Cultice (1984).

Figure 3 illustrates the stimulus material of Homa and Cultice. They are derived from the random 9-dot patterns introduced by Posner & Keele (1968) but Homa has introduced the feature of drawing lines to connect the dots. This makes it relatively cheap to write a computer program that will determine how to map the points of one into another in a way as to achieve maximal fit. Given such a mapping, we can describe each stimulus according to 18 ordered dimensions which are the x and y coordinates of each point. Then we can apply our categorization algorithm to these materials.

Figure 3

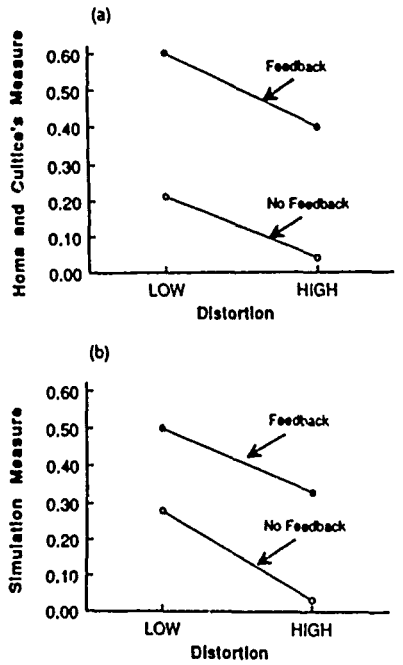


There are three categories in Figure 3--one category represented by 9 items, one by 6, and one by 3. In one condition of their experiment, subjects were given category labels and trained to sort the stimuli into three categories. in another condition they were free to sort the stimuli into whatever categories they wanted. Homa & Cultice were interested in determining how well subjects did at recovering the category structure without feedback. In the case of feedback, Homa & Cultice just measured accuracy of assignment in a final criteria test. In the case of no feedback, they tried to discover some way of assigning labels to the categories in the subjects' sort that made their categorization look optimal. It is hard to know how comparable the two measures are.

In our case, when there was feedback, we measured the probability of a category label according to Equation 1. When there was no feedback, we assigned labels to internal categories in such a way as to maximize probability of a correct label assignment when Equation 1 was used. Again

it is unclear how comparable our two measures were. In our case we corrected our measures for guessing. We ran a control condition where, rather than letting the algorithm decide which items go together, we randomly assigned items to internal categories and then proceeded to get performance scores in the same way as when the algorithm did the assignment. Thus, we got two measures--P, a mean probability of the correct category label when our algorithm did the clustering and G, a mean probability of category labeling in the control condition when we did the clustering randomly. Our final measure was (P - G) / (1 - G) which is a standard correction-for-guessing formula.

Figure 4



Homa and Cultice used a number of different training sets including a low distortion training set where the points were perturbated 1.1 units (the examples in Figure 3 are 1.1 distortions) and a high distortion set where they were perturbated 4.8 units. Figure 4 compares the performance of the subjects and the simulation for high and low distortion training stimuli in the presence of label feedback or not. In the case of Homa & Cultice, we used a correction for guessing measure to but set the guessing rate to be .33 since there were 3 categories. Both subjects and simulations show approximately additive effects of the two dimensions. Both the subjects and the simulation are nearly at chance in the presence of high distortion stimuli with no label feedback. However, our model does show greater sensitivity to feedback.

10. Effects of Input Order In the presence of weak-category structure, the categories the model forms is sensitive to presentation order. In this way we are able to simulate the data of Anderson (1990) and Elio & Anderson (1984).

Comparisons to Cheeseman, Kelly, Self, Stutz, Taylor, & Freeman (1988)

The Bayesian character of this classification model raises the issue of its relationship to the Autoclass model of Cheeseman et al. While it is hard to know how significant the differences are, there are a number of points of contrast:

Algorithm Rather than an algorithm that iteratively incorporates instances into an existing category structure, Cheeseman et al. use a parameter searching program that looks for the best fitting set of parameters. Not enough information is provided to compare the two algorithms with respect to efficiency or probability of identifying the optimal structure. Presumably, Autoclass is independent of the order of the examples.

Number of Classes Autoclass has a bias in favor of fewer classes whereas this bias is setable in the rational model according to the parameter c. Autoclass does not calculate a prior corresponding to the probabilities of various partitionings.

Conditional Probabilities It appears Autoclass uses the same Bayesian model as we do for discrete dimensions. The treatment of continuous dimensions is somewhat different although we cannot discern its exact mathematical basis. The posterior distribution is a normal distribution which will only be slightly different than the t-distribution we use. Both Autoclass and the rational model assume the various distributions are independent.

Qualitatively, the most striking difference is that AUTOCLASS derives a probability of an object belonging to a class whereas the rational model assigns the object to a specific class. However, Cheeseman et al. report that in the case of strong category structure the probability is very high that the object comes from a single category.

Acknowledgments

This research was supported by grant BNS 87-05811 from the National Science Foundation and Contract N00014-90-1489 from the Office of Naval Research.

References

Anderson, J. R. (1990). *The Adaptive Character of Thought.* Hillsdale, NJ: Erlbaum.

Anderson, J.R. & Matessa, M. (In preparation). The Adaptive Nature of Human Categorization. .

Berge, C. (1971). *Principles of Combinatonics*. New York: Academic Press.

Berger, J. O. (1985). *Statistical Decision Theory and Bayesian Analyses*. New York: Springer-Verlag.

Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., & Freeman, D. (1988). A Bayesian Classification System. In *Proceedings of the Fifth International Conference on Machine Learning*. San Mateo, CA: , 54-64.

Elio, R., & Anderson, J. R. (1984). The effects of information order and learning mode on schema abstraction. *Memory and Cognition, 12*, 20-30.

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

Gluck, M.A., & Bower, G.H. (1988). From conditioning to category learning: An adaptive network model. *Journal of Experimental Psychology: General, 8*, 37-50.

Gluck, M.A., & Corter, J.E. (1985). Information and category utility. Unpublished Manuscript. Stanford University.

Hayes-Roth, B. & Hayes-Roth, F. (1977). Concept learning and the recognition and classification of exemplars. *Journal of Verbal Learning and Verbal Behavior, 16*, 321-338.

Hoffman, J., & Ziessler, C. (1983). Objectidentifikation in kunstlichen Begriffshierarcchien. *Zeitschrift fur Psychologie, 194*, 135-167.

Homa, D., & Cultice, J. (1984). Role of feedback, category size, and stimulus distortion in the acquisition and utilization of ill-defined categories. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 10*, 83-94.

Lebowitz, M. (1987). Experiments with incremental concept formation: UNIMEM. *Machine Learning, 2*, 103-138.

Lee, P.M. (1989). *Bayesian Statistics*. New York: Oxford.

Medin, D.L., & Edelson, S.M. (1988). Problem structure and the use of base-rate information from experience. *Journal of Experimental Psychology: General, 117*, 68-85.

Medin, D.L., & Schwanenflugel, P.J. (1981). Linear separability in classification learning. *Journal of Experimental Psychology: Human Learning and Memory, 7*, 355-368.

Medin, D.L, Altom, M.W., Edelson, S.M., & Freko, D. (1982). Correlated symptoms and simulated medical classification. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 8*, 37-50.

Michalski, R. S. & Chilausky, R. L. (1980). Learning by being told and learning from examples: An experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *International Journal of Policy Analysis and Information Systems, 4*, 125-161.

Murphy, G.L., & Smith, E.E. (1982). Basic level superiority in picture categorization. *Journal of Verbal Learning and Verbal Behavior, 21*, 1-20.

Nosofsky, R. (1988). Similarity, frequency, and category represeentation. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 11*, 54-65.

Nosofsky, R.M., Clark, S.E., & Shin, H.J. (1989). Rules and exemplars in categorization, identification, and recognition. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 15*, 282-304.

Posner, M. I. & Keele, S. W. (1968). On the genesis of abstract ideas. *Journal of Experimental Psychology, 77*, 353-363.

Reed, S.K. (1972). Pattern recognition and categorization. *Cognitive Psychology, 3*, 382-407.

Rosch, E., Mervis, C.B., Gray, W., Johnson, D., & Boyes-Braem, P. (1976). Basic objects in natural categories. *Cognitive Psychology, 7*, 573-605.

Search Control, Utility, and Concept Induction*

Brian Carlson, Jerry Weinberg, & Doug Fisher
Department of Computer Science
Box 1679, Station B
Vanderbilt University
Nashville, TN 37235

Abstract

Our research adapts incremental conceptual clustering (or concept formation) to the task of learning to guide search. We build on earlier research that uses concept induction techniques to learn search control, but our approach differs by virtue of its reliance on probabilistic, hierarchical classification schemes that increase certain aspects of search efficiency. The system also includes inductive strategies of 'noise tolerance' that mitigate problems of control knowledge 'utility'. A general lesson is that recently identified search 'utility' problems are synonymous with inductive problems of 'noise'; solutions to the problems of the latter type can be usefully adapted to the former.

1 Introduction

An important objective of machine learning research is to improve the *efficiency* of search. This includes the compilation of operator sequences into macro-operators and the adaptation of object concept learning methods to guide operator application (Mitchell, Utgoff, & Banerji, 1983; Langley, 1985). However, recent research has qualified the naive application of these techniques: learned search control knowledge varies in its *utility* (Minton, 1988). In the worst case, learned knowledge can have a detrimental effect on search since the search to find applicable learned knowledge can be more costly than the search that an uninformed system would require.

This paper illustrates that *incremental conceptual clustering* or *concept formation* can organize search control knowledge for efficient reuse. In particular, operator choices made during successful searches are clustered into 'similarity' classes that capture the shared context in which operators were applicable. Operator selection during later search is guided by classification of contextual information. However, reliance on classification is qualified by 'noise tolerant' strategies that demonstrably mitigate the 'utility' problem. In fact, a general observation of our work is that problems of 'noise' in inductive concept learning and problems of 'utility' in search control learning are closely linked in form and in solution.

*This research was supported by NASA Ames grant NCC 2-645.

2 Search and Concept Induction

There are two facets to the problem of learning search control knowledge: generating plausible abstractions of when operators should be applied and filtering these abstractions for their utility (Etzioni, 1988). These facets correspond to similar aspects of concept learning. This connection has been traditionally recognized with respect to the generation of plausible abstractions. Early work on systems such as SAGE (Langley, 1985) and LEX (Mitchell, Utgoff, & Banerji, 1983) applied empirical concept learning techniques to complete solution traces, thus inducing conditions under which an operator's application previously led to a goal state. Using information-theoretic methods, Rendell, Seshu, and Tcheng (1987) used PLS1, a 'utility' clustering system, to group problems with similar solution strategies. Purely analytic concept learning approaches to uncovering search control knowledge have also been investigated under the rubric of *explanation-based* learning (DeJong & Mooney, 1986; Mitchell, Keller, & Kedar-Cabelli, 1986). These techniques use analytic, typically deductive strategies to find conditions under which 'operators' should apply from a small number of 'solution' traces.

Until recently, work on learning search control has focused almost exclusively on the generation of plausible abstractions However, recently there has been the observation that rule application varies in utility: the degree that rule application alters the number of steps (i.e., subproblems, states) encountered dur-

ing search.[1] Rule application in certain contexts may actually detract from search efficiency. Several approaches to eliminating harmful rule application have been examined. For example, related techniques by Hansson and Mayer (in press) and Wefald and Russell (1989) assess whether significant information is gleaned about eventual goal satisfaction from further state expansion. Probability estimates used in this assessment are learned by analyzing the state space expanded during search. With sufficient experience probability estimates terminate expansion at states from which it is unlikely that useful information can be found by further search (e.g., the system may be very certain about eventual goal achievement from the current state, thus diminishing the need for further search since it is unlikely to yield significant new insights about eventual outcomes).

Recently, research in explanation-based learning has employed similarly-intended, though differently-implemented methods for controlling search (Minton, 1988; Mooney, 1989). Research in this area has focused on the efficacy of exploiting learned rules versus simply using the primitive operators. For example, Markovitch & Scott (1989) learn probability estimates that subgoals can be satisfied. Learned rules are not used in proof attempts of subgoals that are not likely to be successful, only primitive rules are used in these cases, thus avoiding redundant search.

As we have noted, work with SAGE and LEX recognized the applicability of concept induction methods to generate plausible search control rules. Similarly, we believe that utility can be tested by noise-tolerant strategies of concept learning. For example, ID3 (Quinlan, 1986) generates decision trees from training data using a measure of the information transmitted about class membership (e.g., disease) by each attribute used to describe objects (e.g., patient case histories). The values of the most informative attribute label arcs of the tree and are used to divide the training set; the information-theoretic measure is used to recursively divide each training subset, thus forming a decision tree. During tree construction, an estimate of whether 'significant' information is gained about class membership is made by a chi-square heuristic. Similar to Wefald and Russell (1989) and Hansson and Mayer (in press), tree expansion is terminated at nodes where the divisive attribute does not transmit significant information about class membership; at this point, the most common class among the training subset is used to label the appropriate leaf of the decision tree. Subsequent data is classified by traversing appropriate paths of the tree to a leaf, where an appropriate class designation resides. Quinlan and others (Michalski, 1987) have shown that this general strategy eliminates the use of 'low utility' concepts (or portions thereof), classification accuracy is not adversely affected or is actually improved by the process.

In the following two sections we describe the application of empirical concept learning to the induction, organization, and exploitation of search control rules. Like earlier research, we are concerned with the generation of plausible concepts and control rules. Our work in this area is distinguished by the use of *probabilistic* concepts (i.e., control rules), a representation scheme that allows partial matching. Moreover, our work is further distinguished by its attention to the connection between post-generation tests of utility in traditional concept learning systems and search control learning.

3 The COBWEB Concept Formation System

Empirical concept learning is typically concerned with improving prediction accuracy. In search control learning this translates into a concern for accurately predicting the operator that should be applied under current conditions; more accurate predictions result in a more directed, efficient search. For example, the search-intensive task of language recognition is highly anticipatory; a parser expects (i.e., predicts) a particular symbol next on the input stream. If the next symbol is not as expected then the parser has made an incorrect prediction; this may actually reflect on an incorrect prediction that was made several symbols earlier but has only caused a contradiction after subsequent processing. In the case of the phrase *big blue bugle boy*, the subphrase *big blue* might be a nickname, *big blue bugle* might refer to a large blue brass instrument, but the intent of the phrase is that *big*, *blue*, and *bugle* all modify the noun *boy*. Of the relevant parsing operators, (PARSE adjective) and (PARSE noun), neither can be predicted with certainty at intermediate points in the phrase. As with any search-intensive system, the parser must backtrack to an indeterminate depth and try alternatives until contradictions are eliminated.

3.1 Hierarchy Generation

To reduce backtracking we wish to better predict the likelihood that an operator applies under current conditions. Our particular approach to improving search efficiency is through a conceptual clustering system, COBWEB (Fisher, 1987; 1989). This is an unsupervised system that incrementally builds classification trees from objects that are described by nominal attribute-value pairs. Stored at each node of the tree are the value distributions of each attribute over the objects classified under the node. For example, if 90% of the objects stored under a node, n, are blue, then the `blue` feature would be weighted accordingly. $P(\texttt{Color} = \texttt{blue}|n) = 0.9$. Each node is a *probabilistic concept*

[1] Recent work also points out that search control rules vary in their match cost – a test of a rule's applicability (Minton, 1988; Tambe & Rosenbloom, 1989). Our work thus far concentrates on reducing the states examined during search, although we will return to issues of match cost in Section 5.

(Smith & Medin, 1981); the classification tree is a *probabilistic concept tree.*

Each tree level contains sibling classes that collectively partition the observed objects. COBWEB can incrementally incorporate a new object into the class that best matches the object according to *category utility* (Gluck & Corter, 1985), a measure that rewards the formation of object classes that improve 'prediction ability'. More formally, the category utility of a class, C_k, is a function of the *expected number* of attribute values that can be correctly predicted about members of the class, $E(\#CorrectPredictions|C_k)$. This expectation can be further formalized in terms of conditional probabilities that are stored at tree nodes: $E(\#CorrectPredictions|C_k) = \sum_i \sum_j P(A_i = V_{ij}|C_k)^2$. Category utility has some additional complexities (Gluck & Corter, 1985; Fisher, 1987), but for our purposes it is sufficient to note that category utility is a function of:

$$P(C_k) \sum_i \sum_j P(A_i = V_{ij}|C_k)^2,$$

where $P(C_k)$ is the proportion of the observations to which the expectation applies; $P(C_k)$ is the probability that the benefits (i.e., expectations) of a class will be realized.

COBWEB incrementally filters objects into appropriate classes based on category utility. A new object is evaluated with respect to a class by tentatively placing the object in the class; each class's attribute-value distributions are tentatively updated to reflect the values of the new object. Probabilities are computed from the tentatively updated distributions and the category utility of the class is computed. The class that maximizes category utility after adding the new object is chosen to classify the object and appropriate distributions are updated permanently. This process is recursively applied to the subtrees rooted at the selected child until a leaf is reached. A leaf is a singleton class that represents a previously observed object. While objects are predominantly incorporated with respect to existing classes, operators also exist for new node (class) creation, node combination (merging), and node division (splitting). A more complete description of COBWEB can be found in Fisher (1987).

Object incorporation is easily adapted to allow object classification and prediction: category utility guides an object along a path of nodes to a 'best' matching leaf. If any value(s) are missing from the new observation, they may be predicted from the known values of the leaf. While COBWEB trees are reminiscent of decision trees, probabilistic concepts are *polythetic* in that multiple attributes guide classification. If an object has missing attribute values then category utility acts as a partial-matching function with summation limited to probabilities of known attributes.

3.2 Hierarchy Evaluation: Pruning and Utility

COBWEB's strategy of prediction at best matching leaves demonstrably yields good results in many domains, but like early versions of ID3 this strategy can often diminish prediction accuracy (cf., Section 2). More generally, all inductive learning systems require that a domain exhibit regularities (i.e., dependencies between attributes) if learning is to be beneficial. If the learning system is too persistent in trying to uncover regularities where no significant ones exist, then this can result in 'overfitting'. This is also the case in learning to search: if no or little correlation exists between the conditions of a state and the eventual success of an operator application, then persistence in trying to discover such a connection will result in overfitting. In search control tasks, overfitting reduces the accuracy with which operator applications are predicted, thus causing greater backtracking. In both concept learning and search control learning, the utility of certain 'rules' is negligible or detrimental.

A recent version of COBWEB (Fisher, 1989) employs a *past-performance* method for disposing of low utility rules. This method was inspired by Quinlan's (1987) *reduced error* pruning, but the general approach is also related to Hansson and Mayer's and Wefald and Russell's strategies for terminating search.[2] In particular, as COBWEB classifies an object it determines at each node whether an attribute would be correctly predicted at the node. To do this, it compares the object's actual value along this attribute with the most common (i.e., most probable) value of the attribute at the node. If the two values are equal, then COBWEB would have correctly predicted the attribute's value if it had been required to; in this case, a counter is incremented indicating that a correct prediction would have been made at this node. In addition, COBWEB also records whether the attribute would have been correctly predicted at a descendant of the node. Thus, each node holds two counts for an attribute: one of how often a correct prediction would have been made at the node, and one of how often a correct prediction would have been made at a descendent of the node. When a prediction of an attribute is actually necessary (i.e., the attribute's value is unknown), then classification descends to a node at which the unknown attribute is more accurately predicted relative to its descendants, the most common value of the attribute is used as COBWEB's prediction.

Our summary of this past-performance 'pruning'

[2] Fisher (1989) also experimented with a chi-square method of terminating classification that directly assesses the significance of the information gained by deeper classification. Gennari, Langley, and Fisher (1989) used a cut-off parameter to prune a classification subtree based on a user-supplied threshold of required 'information gain'. This method was not sensitive to differences between attributes and relied entirely on the user to specify 'significant' gain thresholds.

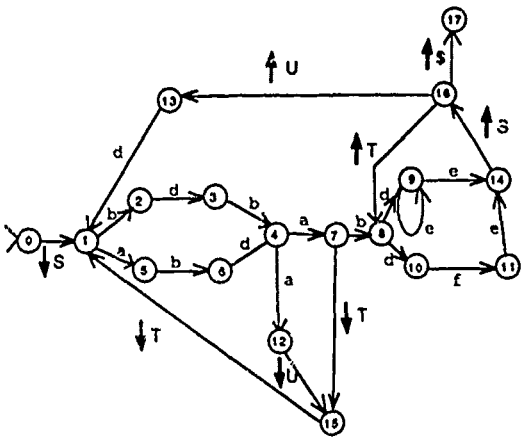


Figure 1: A NPDA transition diagram for simplified-English parsing.

strategy has been brief of necessity, but it does not add to the asymptotic cost of learning or classification and it considerably improves prediction accuracy over the initial strategy of always classifying an object to a leaf. With this in mind, we turn to the application of these concept formation strategies to our primary goal: the effective management of search control.

4 An Example: Search Control in Parsing

Our objective is to demonstrate that COBWEB (and by implication other conceptual clustering systems as well) can effectively direct search by predicting operators that are best applied under 'current circumstances'. Consider a detailed parsing example similar to the one given at the beginning of Section 3. Figure 1 shows a nondeterministic (i.e., backtracking) push-down automata (NPDA) for recognizing an artificial, but nontrivial language. Arcs have two types of labels: those preceded by an up/down arrow, and those that are a lower case letter. The lower case letters denote input symbols that are to be parsed. A down arrow, (↓), followed by a letter denotes a symbol to be pushed on a stack when the arc is crossed in parsing. An arc labeled by an up arrow, (↑), denotes that the symbol must reside at the top of the stack, and is popped from the stack. It is assumed that the stack contains only a $ when parsing begins. A sentence is accepted (i.e., successfully parsed) if there is at least one way to enter the *final state* of the machine (i.e., state 17) with an empty stack and an empty input stream.

Consider "a b d a b b d b a b d e e e e d e", which is a string accepted by the NPDA. A successful parse begins by pushing S on the stack in crossing from state 0 to state 1, parsing "a b d a b" leaving the system in state 7. At state 7, a T is pushed on the stack, then an S leaving the system in state 1. Next the symbols, "b d b a b d e e e e" are parsed, and then the S and T are popped off the stack. Next "d e" are parsed and the final S and $ are popped off the stack, leaving the system in state 17, the final state.

Our example illustrates the moves necessary for a successful parse, but the NPDA contains 4 points of nondeterminism: state 4 given an 'a', state 7 given a 'b', state 8 given a 'd', and state 9 given an 'e'. The arcs out of state 9 model precisely the dilemma of the noun/adjective example at the beginning of Section 3. This nondeterminism is the cause of search: each point of nondeterminism must be tried and may result in backtracking if the wrong choice is made. The NPDA is constructed so that certain incorrect guesses are discovered rather quickly, while others are not uncovered for several moves.

To reduce backtracking we present information about successful parses to COBWEB, in the hope that the system can use this information to guide future parsing. In particular, after a sentence is successfully parsed, a complete trace of the choices *required* from the start state through the final state is returned, this does not include the choices that were retracted via backtracking. Each choice is regarded as an operator to be predicted from decisions that were made previously. Thus, the complete trace is segmented into 'objects' (i.e., windows) of seven attributes. one object for each choice made in the trace. The values of four of these attributes are the four choices (operators) that were made just prior to the current choice; two of these attributes are the top two stack symbols at the time of the current choice; finally, the seventh attribute is the current choice. Thus, a single parse of ten choices will be segmented into ten distinct objects.[3] After sentence recognition, the successful parse is segmented and the constituent objects are incorporated into a COBWEB hierarchy. During subsequent parsing COBWEB is used as an oracle for predicting the current choice, given a window of four previous choices and the top-most stack symbols.

To test the savings provided by a COBWEB oracle, a COBWEB-enhanced parser was trained on successful parses. Training sentences were generated so as to usually adhere to two rules, although neither was followed 100% of the time, thus introducing some 'noise'. The first rule was that pushing a T or a U (leading into state 15) was dependent upon the path from state 1 to state 4. The second regularity was that roughly 4 consecutive *e*'s should occur when in state 9. The COBWEB-enhanced parser was trained on five randomly generated sentences (under the above men-

[3] Note that the initial choices – those without four predecessors – are still represented but without some initial attributes.

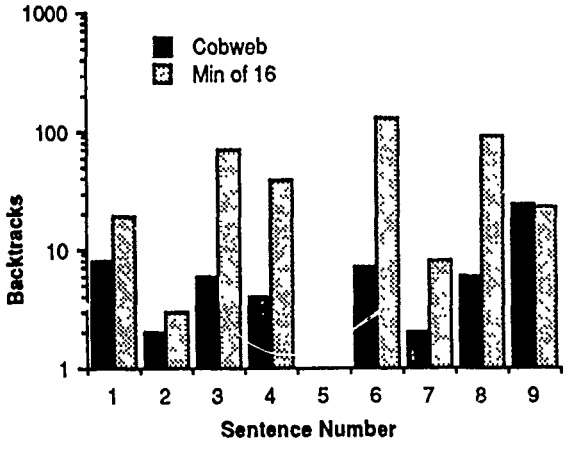


Figure 2: Backtracks required by COBWEB-guided parser and best of 16 alternative parsers.

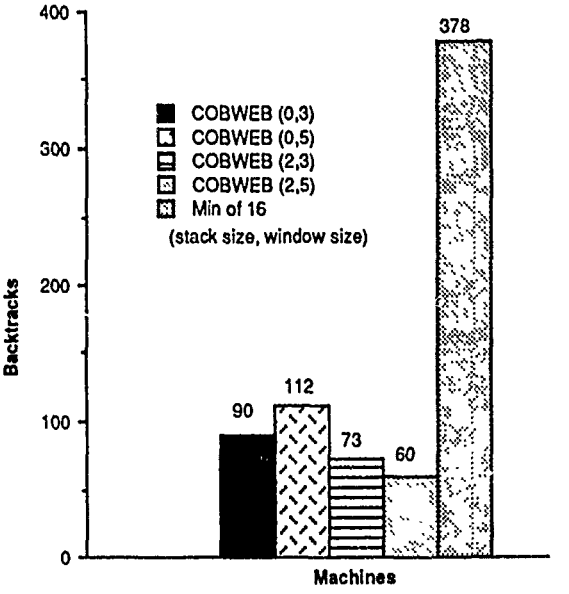


Figure 3: Sensitivity to input window and stack symbols.

tioned regularity constraints), which after windowing constituted a total of one to two hundred training objects. Four more sentences were randomly generated to use as additional test sentences.[4] For comparative purposes, the number of backtracks required by a COBWEB-enhanced parser was compared with sixteen *hard-coded* parsers: recall that the NPDA of Figure 1 has four points of nondeterminism – two choices per point. The sixteen hard-coded parsers correspond to the (2^4 =) 16 possible orderings on these choices. These orderings roughly correspond to all the possible ways that an 'expert' might order choice preferences with sufficient knowledge of individual choice frequencies. All nine (training and test) sentences were run against the 16 hard-coded machines; COBWEB was trained on the first 5 of these sentences and like the hard-coded machines was tested against all nine.

Figure 2 compares the number of backtracks required by the COBWEB-enhanced parser and the number required by the *best* hard-coded machine (per sentence) for each of the nine sentences (note the logarithmic vertical scale). The COBWEB-enhanced parser out performs the minimums of the hard-coded machines on all sentences, except one (sentence # 5) where the number of backtracks was one in each case. A nonparametric sign test reveals that the COBWEB-enhanced parser properly minimized backtracks over the collective minimum of the hard-coded machines in a statistically-significant ($\alpha = 0.01$) number of cases (i.e., 8, with 1 tie); even if we could determine *a priori* the best hardcoded machine to parse each sentence, the enhanced parser would still win in terms of required backtracks. Overall, the COBWEB-enhanced parser requires 60 backtracks for all nine sentences, while the sum of the hard-coded machine minimums comes to 378 backtracks. If we compare the COBWEB-enhanced parser to the individual machines then the savings become even more pronounced: the backtracks required by the individual machines ranged from 643 to 2626, as compared to 378 for their collective minimum and 60 for our COBWEB parser.[5]

We also investigated the sensitivity of the COBWEB-enhanced parser to window size and number of stack symbols that were used during learning. In the results that we report above, we assumed a input window size of 5 and access to the top 2 stack symbols.[6] Figure 3 compares the total number of backtracks over all sentences for various window/stack sizes and the sum total of the minimum parse of each sentence for any of the static machines (Min of 16). For each of

[4]Originally, five additional test sentences were generated, but one was identical to a training sentence and was removed from consideration.

[5]There are several interesting issues that arise when we consider using concept formation techniques to guide language parsing. In particular, the relative performance merits of and conceptual links between our heuristically-guided parser and efficient parsers for such things as LR(k) grammars are of some interest, but are tangential to the objectives of this paper. Notably, our heuristic parser assumes that statistical correlations exist between parsing transitions; if such correlations exist then concept induction techniques can hope to provide some speedup over standard parsing methods, regardless of the language class. If no such correlations exist, then inductive techniques will provide no speedups over the standard parsing methods for a language. More generally, the objectives of this paper are to illustrate links between inductive concept formation and search control management, and thus we will defer discussion of parsing-specific issues.

[6]As we stated earlier, during testing, 4 of the 5 window symbols will correspond to the 4 previous choices, while the 5th will correspond to the choice to be predicted.

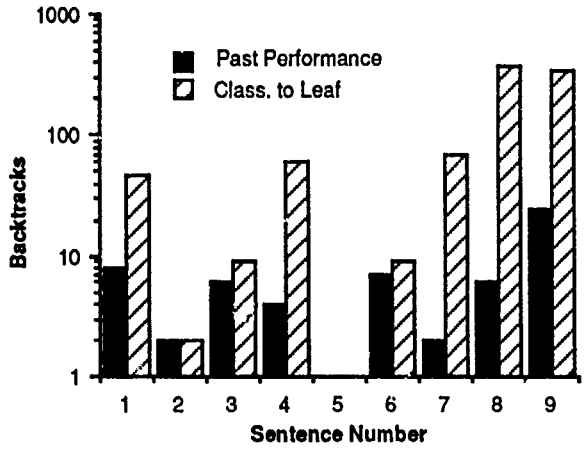


Figure 4: Backtracks required with and without noise-tolerant classification strategy.

the window/stack combinations that we investigated, the number of backtracks required by the COBWEB parser is considerably less than the hardcoded minimums. A second observation is that the performance of our parser appears to improve as the amount of information (input and stack symbols) available to guide classification increases. The exception to this trend occurs between (0,3) and (0,5), but we believe this to be an aberration caused by an exceptional sentence. In fact, the (0,5) properly minimized the backtracks relative to the (0,3) condition for 6 of the 9 sentences, with 1 tie and 2 cases in which (0,3) minimized backtracks. Though our experiments are not sufficient to make statistically-justified claims for the relative advantage of differing (stack, window) conditions at the $\alpha = 0.05$ level, we nonetheless believe that more extensive experiments will confirm the trend within certain limits.

Finally, the COBWEB results reported above assume the classification strategy designed to avoid overfitting that was described Section 3. To test our contention that this strategy avoids overfitting and the detrimental use of low-utility control knowledge, we compared these results to a parser that made predictions suggested by best-matching leaves of the learned concept tree. Figure 4 compares these alternative versions; classification to a leaf yields less reliable results and is often much more costly in terms required backtracking. The past-performance strategy outperforms classification to a leaf in 7 of the 9 cases, with 2 ties. This is significant using the nonparametric sign test at $\alpha = 0.05$. This supports our specific claim that our past-performance strategy gives a good measure of rule utility. More generally, a promising conjecture is that strategies designed to enhance noise tolerance in concept learning may be useful in mitigating utility problems in search control.

5 Concluding Remarks

We have demonstrated the efficacy of concept formation in the management of search control knowledge. COBWEB's probabilistic representation of search control facilitates greater flexibility in guiding search: in addition to hard-and-fast rules, tendencies ('hunches') also guide search. This characteristic is particularly important in tasks like parsing since it may be impossible to tell with certainty whether a particular operator is applicable prior to its invocation.

Although our approach was tested on a parsing task we believe that it can be adapted to other search-intensive tasks, though this may require that we overcome representational limitations. In particular, COBWEB requires nominal attribute-value representations. This representation is sufficient to deal with certain other domains that have been examined in search control research such as the 8-tile puzzle and other games (Wefald & Russell, 1989), but more complicated search tasks such as planning and those found in EBL research will require relational representations. In fact, extensions of COBWEB-like strategies that deal with relational representations are being investigated (Yoo & Fisher, 1990; Yang, Fisher, & Franke, in press).

A second, but more subtle, representation issue arises when we examine more deeply the meaning of rule 'utility'. In particular, we have ignored the 'match-cost' aspect of utility (Tambe & Rosenbloom, 1989); the COBWEB-enhanced parser uniformly requires greater execution time because of increased match cost. In part, this is due to 'uninteresting' factors such as (1) inefficiencies of the COBWEB implementation, which we have not tailored to this domain, and (2) the exceedingly low cost of backtracking with an NPDA. More fundamentally though, it appears that match costs are magnified by probabilistic concepts, which require that we match many attributes of a concept – even those that may be statistically irrelevant to class membership. Fortunately, the same noise-tolerance strategies that identify when an attribute is best predicted are also useful in determining when attributes are irrelevant for classification purposes. Thus, we believe that these strategies suggest a promising path for mitigating the match-cost factor of probabilistic-rule utility (Gennari, 1989).

Finally, we believe that a notable contribution of this work is that it illustrates a link between concept induction strategies for noise tolerance and search control issues of utility: classification/search should terminate at points that do not helpfully inform prediction. In Section 2 we alluded to a point that we more forcefully argue elsewhere (Fisher & Chan, in press; Yoo & Fisher, 1990) – that overfitting is also at the root of the utility problem as it pertains to EBL and domain theory search. Learned rules may represent logically possible, though statistically idiosyncratic connections between patterns of

operational predicates and target consequences. The application/consideration of such rules may actually have a detrimental effect on subsequent problem solving (Mooney, 1989; Markovitch & Scott, 1989). Our current work seeks to unify inductive issues of noise and explanation-based issues of utility in the context of concept formation over problem-solving experience (Yoo & Fisher, 1990) and plans (Yang, Fisher, & Franke, in press).[7] Our work tentatively suggests that EBL mechanisms of *selective retention* (Markovitch & Scott, 1989; Minton, 1988) and *selective utilization* (Markovitch & Scott, 1989; Mooney, 1989)[8] are synonymous in spirit to, and better implemented by, much studied inductive methods of *pruning* (Quinlan, 1986; Michalski, 1987) and preference identification (Fisher, 1989), respectively.

Acknowledgements

We thank the reviewers for helpful suggestions on content and exposition.

References

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

Etzioni, O. (1988). Hypothesis filtering: a practical approach to reliable learning. *Proceedings of the Fifth International Conference Machine Learning.* (416–428). Ann Arbor, MI: Morgan Kaufmann.

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

Fisher, D. H. (1989). Noise-tolerant conceptual clustering. *Proceedings of the International Joint Conference Artificial Intelligence* (pp. 825–830). Detroit, MI: Morgan Kaufmann.

Fisher, D., & Chan, P. (in press). Statistical guidance in symbolic learning. *Annals of Mathematics and Artificial Intelligence.*

Gennari, J (1989). Focused concept formation. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 379–382). Ithaca, NY: Morgan Kaufmann.

Gennari, J., Langley, P., & Fisher, D. (1989). Models of concept formation. *Artificial Intelligence, 40*, 11–62.

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

[7]Unification appears to be a similar, though implicit, intention behind the work of Etzioni (1988).

[8]We believe that these terms were coined by Markovitch and Scott.

Hansson, O., & Mayer, A. (in press). Probabilistic heuristic estimates. *Annals of Mathematics and Artificial Intelligence.*

Langley, P. (1985). Learning to search: from weak methods to domain-specific heuristics. *Cognitive Science, 9*, 217–260.

Markovitch, S. & Scott, P. (1989). Information filters and their implementation in the syllog system. *Proceedings of the International Workshop on Machine Learning* (404–407). Ithaca, NY: Morgan Kaufmann.

Michalski, R. (1987). How to learn imprecise concepts: a method for employing a two-tiered knowledge representation in learning. *Proceedings of the Fourth International Workshop on Machine Learning* (50–58). Irvine, CA: Morgan Kaufmann.

Minton, S. (1988). Qualitative results concerning the utility of explanation-based learning. *Proceedings of the Seventh National Conference on Artificial Intelligence* (pp. 564–569). St. Paul, MN: Morgan Kaufmann.

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

Mitchell, T , Utgoff, P., & Banerji, R. (1983). Learning problem solving heuristics by experimentation. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.) *Machine Learning: An Artificial Intelligence Approach*, Palo Alto, CA: Morgan Kaufmann.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81–106.

Quinlan, J. R. (1987). Simplifying decision trees. *International Journal of Man-machine Studies.*

Rendell, L., Seshu, R., & Tcheng, D. (1987). More robust concept learning using dynamically-variable bias. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 66–78). Irvine, CA: Morgan Kaufmann.

Smith, E. E., & Medin, D. L. (1981). *Categories and concepts.* Cambridge, MA: Harvard University Press.

Tambe, M. & Rosenbloom, P. (1989). Eliminating expensive chunks by restricting expressiveness. *Proceedings of the International Joint Conference Artificial Intelligence* (pp. 731–737). Detroit, MI: Morgan Kaufmann.

Wefald, E. & Russell, S. (1989). Adaptive learning of decision-theoretic search control knowledge. *Proceedings of the International Workshop on Machine Learning* (408–411). Ithaca, NY: Morgan Kaufmann.

Yang, H., Fisher, D., & Franke, H. (in press). Improving planning efficiency by conceptual clustering. *The Third International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems.* Charleston, SC: ACM Press.

Yoo, J., & Fisher, D. (1990). Concept formation over explanations and problem-solving experience. Technical Report. Department of Computer Science, Vanderbilt University.

Graph Clustering and Model Learning by Data Compression

Jakub Segen
AT&T Bell Laboratories, rm. 4E-632
Holmdel, N.J. 07733

Abstract

This paper uses a minimal representation criterion to formally define tasks of matching, classification, and interpretation of objects represented as graphs, as well as conceptual clustering of graphs, and supervised learning of structured concepts represented as probabilistic graphs. These definitions do not rely on acceptance thresholds, or other user selectable parameters. The resulting problems of combinatorial optimization are approximately solved by a fast graph matching heuristic, which is a key element of the described learning methods, that include forced learning of graph models, and two graph clustering methods: incremental, and agglomerative. These methods apply to usual directed graphs, and to recursively defined layered graphs. The presented methodology has been applied to real problems of concept learning, classification and interpretation of nonrigid shapes.

1 Introduction

Structural descriptions composed from parts and relations are often used to describe complex entities. They have been particularly useful in computer vision, as representations of shape of objects (Barrow et al., 1972; Connel & Brady, 1985; Shapiro, 1980). A convenient form of a structural description is an attributed graph, i.e. a graph with symbolic attributes, or labels, attached to its vertices and edges.

The use of a graph as representation for data and concepts, or models demands formulation of methods for matching data to concepts, object classification, and concept learning. Variety of such methods have been proposed, some based on exact matching, e.g. Winston, (1975), other allowing inexact match by defining a graph distance, or probability, as Wong & You (1985). However, a practical use of many of the graph based methods seems to be limited by computational cost of graph matching, and lack of robustness. The latter is most often caused by use of control parameters, such as acceptance level, threshold, or weights, without automatic methods of their selection, so they may have to be set by the user differently for each application.

Recently, we introduced a method for supervised learning of graph models (Segen, 1988a), which uses a minimal representation criterion (Segen & Sanderson, 1979; Segen, 1980) to define graph matching. This method is very practical, since it is free of user settable parameters, and fast, due to an efficient graph matching technique. It has been applied to recognition and supervised learning of nonrigid shapes.

In this paper we describe methods for graph clustering, based on the same approach, and present general methodology for supervised and unsupervised learning, classification and interpretation, using layered graph representation. The minimal representation criterion is used to define these tasks formally, and to guide their solutions, judging each step by its data compression ability. The presented methods have been implemented, and applied to shape data from real noisy images. They appear quite fast and robust. At the end of the paper we show some examples from this application.

2 Layered graph

A directed graph is a set of vertices V and a set of directed edges E. Beginning with a directed graph we recursively define a special case of a hypergraph called a *layered graph*. A vertex $v \in V$ will be called a level 0 vertex, or a *leaf*. Two level 0 vertices connected by an edge will be called a level 1 vertex. If we assume a set of directed edges over the set of level 1 vertices, we can similarly define a level 2 vertex, and generally define a level n+1 vertex as an ordered pair (v_1, v_2), where v_1 and v_2 are level n vertices. Their order corresponds to the direction of the edge between v_1 and v_2. We will call such a structure a layered graph. A layered graph

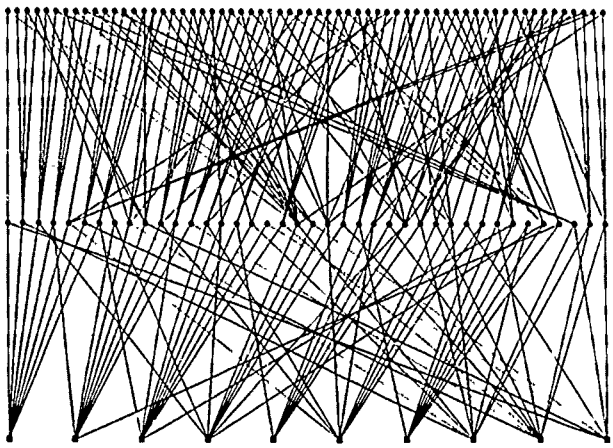


Figure 1: Layered graph. Leaves are at the bottom

can be represented by a directed acyclic graph, whose vertices correspond to the vertices of the layered graph, and edges show the hierarchical dependency among vertices. Figure 1 shows an example of a layered graph used as a shape description. Referring to vertices of a layered graph we will use terms *parent, child, ancestor, descendant* and *leaf*, in the same sense as for a tree, except that here a vertex can have multiple parents.

A layered graph has the following properties:

1. Each non-leaf vertex v has exactly two ordered *children*, distinguished as $left(v)$ and $right(v)$.
2. For every vertex v, every path between v and a leaf vertex have the same length. This length is called the *level* of v.
3. Two vertices can have no more than one common parent.

In addition, we assume that each vertex v of a layered graph has a label $L(v)$, which is a symbol from a finite alphabet, and there is a separate alphabet for each level. Such a graph is a special case of an attributed hypergraph: the leaves of the layered graph are vertices of a hypergraph, while the higher level vertices are the hyperedges. Further we often refer to the layered graph simply as graph.

3 Probabilistic graph

A group of layered graphs that are not identical can be described using a probability model whose outcome is a layered graph, or a *probabilistic layered graph*. We define this model just like a layered graph, except that each of its vertices is associated with a probability distribution over a set of labels, rather than a single label. A probability of finding the label l at a vertex v will be denoted $p(l|v)$,

If T is a mapping from vertices of a probabilistic graph M onto vertices of H, we can assign to each mapped vertex of M the label, of the corresponding vertex of H. The probability of H, given M and T, $P(H|T,M)$ is the probability of this set of assignments. Assuming independently distributed vertex labels, $P(H|T,M)$ is simply the product of the probabilities $p(L(Tv)|v)$, where v is a vertex of M, and Tv is the vertex of H assigned to v by T.

4 Minimal representation criterion

The minimal representation criterion (Segen & Sanderson, 1979; Segen, 1980) was introduced to guide inference of models in cases when the maximum likelihood fails. Its formulation has been inspired by the pioneering work of Solomonoff (1964). The minimal representation criterion seeks, in a given set of programs, a minimal length program generating observed data X. Mapping a family probability distributions to a subset of programs, and seeking a distribution corresponding to a minimal program, results in an inference rule that select a distribution P that minimizes

$$C(P) - \log_2 P(X) \qquad (1)$$

where $C(P)$ is is the number of bits needed to specify the probability model P, or the *cost* of P. This criterion is obviously equivalent to the minimal description length principle of Rissanen (1978, 1987), and related to the information measure of Wallace & Boulton (1968), but it was derived independently.

5 Matching graphs

A key mechanism of a learning system based on a graph representation is a method for establishing a mapping between elements of two graphs, or graph matching. Based on the minimal representation criterion, we define the task of matching a probabilistic graph M to a graph H, as a construction of a mapping between vertices of M and H, that maximally simplifies description of H, when H is represented relative to M. This definition provides a natural decision rule for accepting a match, which does not rely on an arbitrary threshold. Intuitively, it works as follows: If part of M fits to a part of H, then representing H relative to the matching part of M should cost less than its *default* representation, independent of any model. However, the total cost of representing H with the aid of M includes an overhead associated with specifying the mapping. If the part of H represented by M is too small, the saving may not cover the overhead cost, and it will be cheaper to use the default representation, i.e., to reject the match.

A default representation of a graph is constructed as follows:

1. Specify N, the number of leaves in H.
2. Provide a list of leaf labels, ordered by leaf index.

3. Order pairs of leaves (e.g. lexicographically), and specify the label of the common parent for each pair (NIL if none).
4. For level 1, order pairs of non-null vertices. For every pair, specify the label of the common parent (or NIL).
5. Repeat the last step for each higher level, up to a level with all null vertices.

The cost of this representation $C(H)$ is

$$C(H) = C(N) - \sum \log p(L(v)) \qquad (2)$$

where $p(l)$ is a prior probability of the label l. The first term is the cost of representing N, the second term is the cost of specifying vertex labels (this value is within 1 bit from the length of Shannon block encoding).

A graph H can be described relative to a model M, given a one-to-one mapping T from $V_1 \subseteq V(M)$ onto $V_2 \subseteq V(H)$. To represent H under this mapping we use the probability defined by M for labels of the mapped vertices in V_2, instead of their prior probability. The cost of representing H, given M and T, denoted $C(H|T,M)$ is:

$$C(H|T,M) = C(H) - \sum_{v \in V_1} \log_2 \frac{p(L(Tv)|v)}{p(L(Tv))} \qquad (3)$$

where the second term is the sum of bits saved over V_2. The cost of describing H relative to M, $C(H,T|M)$ includes the overhead for specifying T.

$$C(H,T|M) = C(H|T,M) + C(T) \qquad (4)$$

where $C(T)$ is the cost of T. To find this cost, notice that mapping T is completely determined from its submapping T' restricted to the sets of leaves $F(M)$ and $F(H)$. T' is a mapping from $F_1 \subseteq F(M)$ onto $F_2 \subseteq F(H)$, such that $T'(v) = T(v)$ if v is a leaf. This fact that T' determines T is implied by the property 3 of the layered graph (Section 2). Therefore, we only need to specify the leaf mapping T', and $C(T) = C(T')$. If T' maps k out of n leaves of one graph, to k out of m leaves of the other, then knowing n and m, we can encode T' with

$$C(T') = \log_2 \min(n,m) + log_2 \frac{n!}{(n-k)!k!} + log_2 \frac{m!}{(m-k)!} \qquad (5)$$

bits. The first term is the cost of specifying $k > 0$, the second term is the cost of selecting k leaves of the first graph, and the third is the cost of selecting k leaves of the second graph and specifying their permutation. This representation essentially assigns equal prior probability to each value $k > 0$. An alternative is to assume equal prior probability for each mapping, which corresponds to a constant cost

$$C(T') = \log_2 \sum_{k=1}^{\min(n,m)} \frac{n!m!}{(n-k)!(m-k)!k!} \qquad (6)$$

We prefer the first representation, since it is less penalizing to small values of k. The number of bits saved with respect to the default representation is

$$\begin{aligned} Q(H,T|M) &= C(H) - C(H,T|M) \qquad (7) \\ &= \sum_{v \in V_1} \log_2 \frac{p(L(Tv)|v)}{p(L(Tv))} - C(T) \end{aligned}$$

If this value is positive, the representation based on M and T should be used instead of a default representation, since it is less expensive.

We define the task of matching a graph H with a model M as a problem of constructing a mapping T, that maximizes the value of $Q(H,T|M)$. It is easy to show that the problem of finding maximal isomorphic subgraph, which is NP-complete, is a special case of the above problem, so the maximization of $Q(H,T|M)$ is NP-hard. Therefore, we must compromise and accept a quasi-optimal solution obtained by heuristic search. The fast graph matching heuristic described below is an improved version of the method from Segen (1987).

The procedure for matching H with M consists of two steps, called *map* and *refine*. *Map* finds an initial mapping T that maximizes an upper bound of Q, then *refine* iteratively edits T until Q reaches a local maximum.

The function *map* uses contextual similarity as a basis for the leaf assignment. We can think of a vertex v of a layered graph as a relation $l(f_1, f_2, \ldots f_k)$, where l is the label of v, and $f_1, f_2, \ldots$ are recursively ordered leaf descendents of v. Similarly, we can associate with a vertex v of a probabilistic layered graph a relation $l(f_1, f_2, \ldots f_k)$, with a probability $p(l|v)$. We consider the context of a leaf v to be the set of relations associated with its ancestors. It is described by a *support* of the leaf v, denoted $SPS(v)$. A support of a leaf v of a layered graph, is a set of pairs $\{(R,K)\}$, where R describes a relation by its label and the position of the vertex v in its argument list, and K is the number of occurrences of R among ancestors of v. A support of a leaf v of a probabilistic layered graph, is a set of pairs $\{(R,G)\}$, where R describes a relation, as above, and G is a list containing a value $2^{-level(u)} \log \frac{p(l|u)}{p(l)}$ for each occurrence of R in an ancestor u of v, for which $\frac{p(l|u)}{p(l)} > 1$. This list is sorted in a decreasing order. The factor $2^{-level(u)}$ divides equally the support of u among its leaf descendents.

A *similarity* $S(u,v)$ between a leaf u of a graph, and v is a leaf of a probabilistic graph is defined as

$$S(u,v) = \sum_{\substack{(R,K) \in SPS(u) \\ (R,G) \in SPS(v)}} \sum_{i=0}^{\min(K,|G|)} G_i \qquad (8)$$

$S(u,v)$ is an upper bound on an increase of Q resulting from adding an assignment (u,v) to T.

Map finds a mapping that maximizes the sum of similarities between mapped leaves, by solving an assignment problem. This mapping is iteratively improved by *refine*, which deletes or adds one assignment in each iteration, seeking a maximal increase of Q. The iteration stops when no single addition or deletion can further improve Q.

6 Recognition and interpretation

Given a library of probabilistic graph models $ML = \{M_0, M_1, M_2, ...M_k\}$, and a graph H, we might want to select one model M, which is in some way nearest to H. We call this task *recognition*, or *classification*, since each model from ML is associated with a class of graphs for which it is nearest. Since the mapping of H to a single model may not exhaust all the vertices of H, we may want to map parts of H to several models. These mappings will form an *interpretation* of H. Interpretation can be considered an attempt to explain H using the models from from the library as primitives. Since both recognition and interpretation can result in a compressed representation of H, we define both tasks as problems of minimizing representation cost.

6.1 Recognition

Recognition is a problem of finding a model m in ML, and a mapping T from m to H, that maximizes

$$Q(H,T|m) + \log_2 P(m) \qquad (9)$$

Here $P(m)$ is the prior probability of model m, and we assume M_0 to be an empty graph always associated with a null mapping, so that $Q(H,T|M_0) = 0$. A simple brute force recognition method seeks a maximum of this expression by matching H with each model in ML. The computational cost of this method grows approximately linearly with the size of the model library. We are presently experimenting with screening methods based on bounds provided by the similarity function (Equation 8) to speed up the recognition for large libraries.

6.2 Interpretation

Interpretation is a problem of constructing a sequence of models from ML, $m_1, m_2, ...m_k$, where $m_k = M_0$ and $m_l \neq M_0$ for $l < k$, and a sequence of associated mappings $T_1, T_2, ...T_k$ which maximize the value of

$$\sum_{i=1}^{k} Q(H_i, T_i|m_i) + \log_2 P(m_i) \qquad (10)$$

The graphs H_i, $i = 1,2,...k$ in the above expression are constructed recursively, by taking $H_1 = H$, and forming H_{n+1} from H_n, by removing all vertices of H_n mapped to m_n by T_n. A heuristic method for graph interpretation solves the recognition problem, first for $H_1 = H$, then for H_2, etc., until some graph H_k is recognized as M_0. Since interpretation requires multiple recognitions, it will also benefit from the use of screening in the recognition search.

7 Learning

Models used for recognition and interpretation can be learned from a training set of graphs. If the graphs in the training set are grouped into classes, and our objective is to find models that correspond to these classes, we have a task of supervised learning. We can approach it in two ways. First, we can force a single model to represent all graphs within a class. However, such a model may not perform well if the graphs in a class are not sufficiently similar. In such a case, we can try to divide a class into subclasses containing mutually similar graphs, and use a separate model for each subclass. This is an unsupervised learning task.

Below we describe a method of forced learning of a single class model (Segen, 1988a), and new clustering methods for unsupervised learning.

7.1 Forced learning

This learning procedure incrementally constructs a probabilistic graph for a class, from the sequence of its members. Given a sequence of graphs $H_1, H_2, ...H_k$, the first graph H_1 is converted to a probabilistic graph M_1, by assigning a probability value to each vertex label, using the formula below. The following graphs are then used to update the model, which results in a sequence of models $M_1, M_2, ...M_k$, using the following match-and-merge operation.

A graph H_{n+1} is matched with model M_n, giving a mapping T. Since T maps a subset of leaves of M_n to a subset of leaves of H_{n+1}, generally, some leaves of M_n and H_{n+1} remain unmapped. The mapping T and the graph M_n are then extended to T' and M' in the following way: Initially M' is set to M_n. For each unmapped leaf f of H_{n+1}, a new leaf f' is added to M', and a pair (f', f) is added to T. Then new higher level vertices are added to M', such that each vertex of M' has a corresponding vertex in H_{n+1}. The mapping of these vertices is determined by recursively following child to parent links. Finally, vertex-label statistics for the vertices of M' are updated, based on the labels of their matches in H_{n+1}, and new label probabilities are computed using the Bayes estimator: $P(i) = \dfrac{n(i)+1}{n+k}$, where $n(i)$ is the number of events in the i-th category, n is the total number of events, and k is the number of categories. The final form of graph M' is then taken as M_{n+1}. The last graph M_k obtained this way is used as the model of a class.

7.2 Graph clustering

While most of the work on conceptual clustering is focused on attribute-value, or vector representations, (Cheeseman et al., 1988; Fisher, 1987a, 1987b; Fisher & Langley 1985; Michalski & Stepp 1983; Segen & Sanderson, 1979; Segen, 1980, 1988b, 1989; Wallace & Boulton, 1968), several methods have been proposed for clustering graphs (Levinson, 1984; Wong & You, 1985;) or structured entities that can be represented by graphs (Stepp & Michalski 1986; Lebowitz, 1987; Thompson & Langley, 1989; Wogulis & Langley, 1989). These methods share the requirement for a user specified parameters or other subjective devices to control the number of clusters, or cluster separation. Here we attempt to formulate the clustering of graphs in a non ad hoc manner, as an optimization problem void of any free parameters.

Graph clustering is a task of dividing a set of graphs into groups, such that similar graphs are grouped together. We can formally define such a task as a problem of minimizing the cost of representation of a set of graphs. Given a sequence of graphs $I = H_1, H_2, ...H_N$, we want to construct a set of probabilistic models, $ML = M_1, M_2, ...M_k$, such that when graphs in I are represented relative to models in ML, the total cost of representing ML, and I is minimum. The set of all graphs that are represented relative to the same model is called a cluster. To avoid the need for encoding label distributions for probabilistic graphs we will represent a model predictively, using a small set of cluster members. They will be called *internal* members, while remaining members of the cluster will be called *external*. This approach is related to Rissanen's predictive minimal description length principle (Rissanen, 1987), but it does not depend on ordering of data elements.

A model M_i, $i > 0$, will be represented by a *generating* sequence of n_i graphs from I, $I_i = H_{i1}, H_{i2}, ...H_{in_i}$ (internal members). The forced learning algorithm applied to this sequence results in a sequence of models $M_{i1}, M_{i2}, ...M_{in_i}$. The last model in this sequence is then used as the model M_i of a cluster i. We use a default representation for the first graph in the generating sequence, and then encode the following graphs predictively, i.e., a graph H_{ij+1} is represented relative to a model M_{ij}. In addition, for each graph in I_i we must encode its position in I, to preserve the initial ordering of I.

The total representation for ML and I consists of the following parts:

1. The length N of I, and the number of clusters K.
2. The predictive encoding of a generating sequence for each cluster.
3. The position of each of $n = \sum n_i$ internal members in I.
4. For each external member H of some cluster, a cluster index i, and the representation of H relative to M_i.
5. For each *free* (not a member of any cluster) graph H, the index of the group of free graphs, and the default representation of H.

An incremental clustering method called INCLAG, begins by forming a cluster from the first element of I, then it assigns each successive element to its nearest cluster, or creates a new cluster containing this element, based on the value of expression (9). The nearest cluster is the one which gives maximal value to this expression; a new cluster is formed if this value is not positive. After examining the last element of I, all singleton clusters are eliminated, and their members become free graphs.

While INCLAG is simple and fast, its results depend on the order of data, and it tends to finds only well separated clusters. A more complicated graph clustering method called ACLAG, uses an agglomerative procedure, which does not dependent on order of data. Comparing to INCLAG, it can better separate similar clusters, but it is also more expensive. Beginning with a default representation for each graph in I, and 0 clusters, ACLAG repeatedly applies one of the following moves, until there is a single cluster containing all elements of I:

1. Form a new cluster from a pair of free graphs.
2. Assign a free graph to one of the clusters, as an external member.
3. Merge two clusters by assigning members of the first cluster to the second cluster as external members, and removing the first cluster.

Each iteration selects the move, which results in a minimal representation cost after the move, even if this cost increases. When a new member is added to a cluster, ACLAG attempts to reduce its cost by appending external members to the generating sequence. The final result is the best among the examined sets of clusters.

8 Application example: Learning shape models

The methods presented in this paper have been applied to modeling, recognition and interpretation of planar shapes. Details of this application will be described in a separate paper, and here we show only several examples to illustrate the methodology, and give a feel of its practical potential.

A general task is to learn models of shape classes from a training set, consisting of shape examples and their class names, and to use these models to explain new simple or composite shapes. To represent shape in a layered graph structure we need to define primitive parts that will form leaves of the graph, and relations, to be represented by higher level vertices. In

our representation (Segen, 1988a), the primitives, represented by leaves, are points of extremal curvature of the contour, along with their local tangent and curvature. Higher level vertices represent geometric relations among the primitives, based on distances and angles: First level correspond to binary relations, second level vertices to fourth order relations, etc. Figure 2 shows a shape with labeled points of extremal curvature (leaves), and labeled binary relations, i.e. the first two layers of a graph. A complete three layer graph representing this shape (without vertex labels) is shown in Figure 3.

In one application we used 5 classes of such shapes. Figure 4 shows one example of each of these classes. The training set consisted of 270 examples (54 per class), and the test set of 146 shapes. When we applied the forced learning method and used the resulting models to recognize test examples, 105 examples (about 72%) were recognized correctly. After grouping the training examples by incremental clustering (INCLAG), the number of correctly recognized cases increased to 127 (about 87%).

To see how automatic clustering relates to our subjective classification, we applied the agglomerative clustering method (ACLAG) to a mixed collection of training examples. The result in Figure 5 shows that classes are subdivided with only few misplacements.

9 Conclusion

This paper applied the criterion of minimal representation to formulate graph matching, classification, interpretation, and learning probabilistic graph models as combinatorial optimization problems. The methods proposed to approximately solve these problems rely on an efficient graph matching heuristic. These methods are quite practical, and have been applied to recognition and interpretation of nonrigid shapes using real, noisy data. Possible extensions of this work may include other types of graphs and hypergraphs, in particular undirected graphs. However, more immediately needed is the analysis accuracy and speed of presented heuristics.

9.1 Acknowledgements

I would like to thank Kristin Dana for her effort in collecting the shape data. I also thank reviewers of this paper for their comments and suggestions.

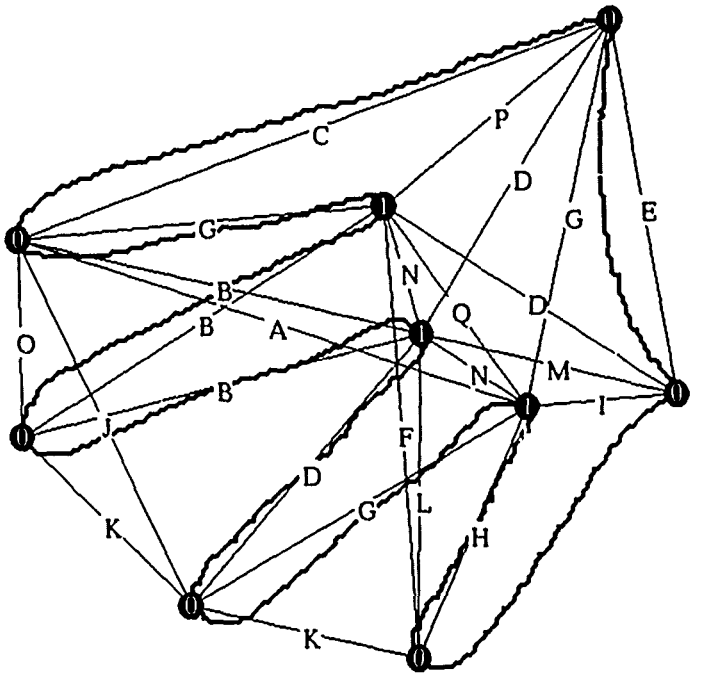


Figure 2: Structural representation of shape. Circles correspond to leaves, thin straight lines between circles represent binary relations (1st level vertices). Numbers and letters are vertex labels.

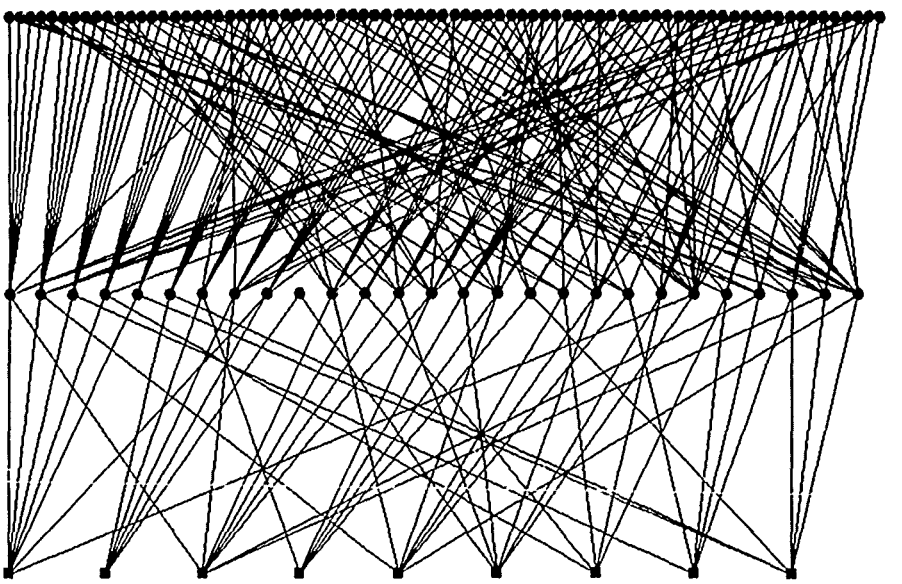


Figure 3: Layered graph representing the shape from Figure 2.

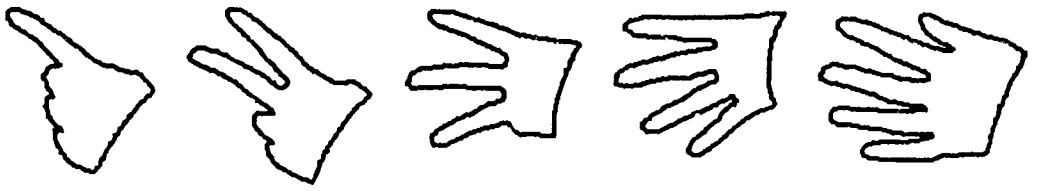


Figure 4: Examples of 5 shape classes.

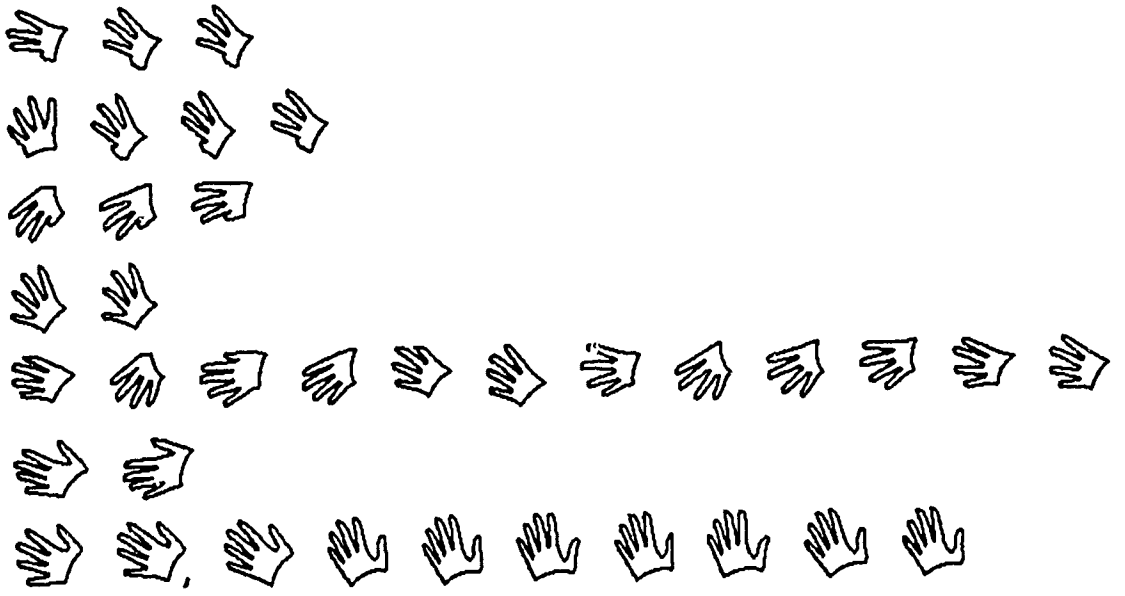


Figure 5: Shape clusters from ACLAG. Each row shows members of one cluster.

References

[1] Barrow, H. G., Ambler, A. P., & Burstall, R. M. (1972). Some techniques for recognizing structure in pictures. In S. Watanabe (ed.) *Frontiers of Pattern Recognition* 1-29. New York: Academic Press.

[2] Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., Freeman, D. (1988). AutoClass: A Bayesian Classification System. *Proceedings of 5-th International Conference on Machine Learning,* 54-64. Ann Arbor, MI, June 1988.

[3] Connel, J. H., & Brady, M. (1985). Learning shape descriptions. In *Proceedings of the Ninth International Conference on Artificial Intelligence,* 922-925. Los Angeles, CA: Morgan Kaufmann.

[4] Fisher, D. H. (1987a). Conceptual Clustering, Learning from Examples, and Inference. *Proceedings of the 4-th International Workshop on Machine Learning* 38-49. University of California, Irvine, June 1987.

[5] Fisher, D. H. (1987b). Knowledge Acquisition via Incremental Conceptual Clustering. *Machine Learning 2,* 139-172, 1987.

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

[7] Lebowitz, M. (1987). Experiments with incremental concept formation: Unimem. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchel, (Eds.) *Machine Learning:2,* Los Altos, CA: Morgan Kaufmann.

[8] Levinson, R. (1985). *A self-organizing retrieval system for graphs.* Ph.D. Thesis, University of Texas, Austin, TX.

[9] Michalski, R. S., & Stepp, R. E. (1983). Learning from observation: conceptual clustering. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchel, (Eds.) *Machine Learning,* Los Altos, CA: Morgan Kaufmann.

[10] Rissanen, J. (1978). Modeling by shortest data description. *Automatica 14,* 465-471.

[11] Rissanen, J. (1987). Stochastic complexity. *J. Royal Stat. Soc. vol. 49* 223-239.

[12] Shapiro, L. G. (1980). A structural model of shape. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-2,* 111-126.

[13] Segen, J., & Sanderson, A. C. (1979). A minimal representation criterion for clustering. *Proceedings of 12 Annual Symposium on Comp. Science and Statistics,* Univ. of Waterloo, Canada, May 1979.

[14] Segen, J. (1980). *Pattern-directed signal analysis.* Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA.

[15] Segen, J. (1988a). Learning graph models of shape. *Proceedings of the 5-th International Conference on Machine Learning,* 29-35. Ann Arbor, MI, June 1988.

[16] Segen, J. (1988b). Conceptual Clumping of Binary Vectors with Occam's Razor. *Proceedings of 5-th International Conference on Machine Learning,* 47-53. Ann Arbor, MI, June 1988.

[17] Segen, J. (1989). Incremental Clustering by Minimizing Representation Length. *Proceedings of the 6-th International Workshop on Machine Learning,* 400-403. Cornell University, Ithaca, New York, June 1989.

[18] Solomonoff, R. J. (1964). A formal theory of inductive inference I, and II. *Information and Control, 7,* 1-22 & 224-254.

[19] Stepp, R. E., & Michalski, R. S. (1986). Conceptual clustering: Inventing goal-oriented classifications of structured objects. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchel, (Eds.), *Machine Learning, (Vol. 2).* Los Altos, CA: Morgan Kaufmann.

[20] Thompson, K., & Langley, P. (1989). Incremental Concept Formation with Composite Objects. *Proceedings of the 6-th International Workshop on Machine Learning,* 371-374. Cornell University, Ithaca, New York, June 1989.

[21] Wallace, C. S. & Boulton, D. M. (1968). An information measure for classification. *Comp. Journal 11(2),* 185-194.

[22] Winston, P. H. (1975). Learning structural descriptions from examples. In P. H. Winston (Ed.), *The Psychology of Computer Vision,* (Chapter 5). New York, NY: McGraw Hill.

[23] Wogulis, J., & Langley, P. (1989). Improving efficiency by learning intermediate concepts. *Proceedings of the 11-th International Conference on Artificial Intelligence,* 657-662. Detroit, MI: Morgan Kaufmann.

[24] Wong, A. K. C., & You, M. (1985). Entropy and distance of random graphs with application to structural pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-7,* 599-609.

CONSTRUCTIVE INDUCTIONAND REFORMULATION

An Analysis of Representation Shift In Concept Learning

William W. Cohen
Computer Science Department
Rutgers University
New Brunswick, NJ 08903

Abstract

In spite of the importance of representation in learning, little progress has been made toward understanding what makes representations work. This paper describes a framework for knowledge-level analysis of changes in the representation of training examples in concept learning. This a very fundamental sort of representation change; such a change alters the very space over which learning occurs, and hence necessitates selection of a new hypothesis space and (probably) a new learning algorithm. The goals of this paper are first, to provide a framework for analysis of representation shifts; second, to make explicit the assumptions implicit in representation shifts that have actually been used in learning systems; and third, to suggest a procedure for finding the most appropriate representation shift, given some background knowledge about a learning problem. The analytic framework is used to analyze a class of hybrid EBL/SBL systems by characterizing the sorts of domain theories that can be used with these systems.

1 Introduction

Few will take issue with the claim that the choice of an appropriate representation is crucial to success in solving AI problems. This is particularly true for learning problems. However, in spite of the importance of representation in learning, little progress has been made towards *understanding* what makes representations work: determining whether a representation will or will not help in solving a learning problem is still by and large a black art. In particular, there is no accepted procedure for answering questions about representation change such as: when is a representation appropriate for a learning problem? what assumptions are being implicitly made when a particular representation is being used? given some background knowledge about a learning problem, which representations are good, which are bad, and which are optimal?

The subject of this paper is *representation change in learning*; in particular, changes in the representation of training examples in concept learning problems. This is in contrast to previous work in representation change in learning, which has primarily addressed the problem of changing the representation of hypotheses (*i.e.*, shift of inductive bias.) Since a hypothesis is a set of examples, a change in the representation of training examples requires change in inductive bias; however, being able to change the language used to describe examples *as well as* the language used to describe hypotheses makes possible more radical representation shifts.

The problem of automatically performing representation shifts of this kind is outside the scope of this paper. The goals of this paper are:

1. to provide a framework for analysis of representational choices made by humans,
2. to uncover the assumptions implicit in certain representation shifts that are used in real learning systems, and
3. to suggest a methodology for selection of an appropriate representation shift, given some background knowledge about the learning problem.

All analysis is done at the "knowledge level" [Newell, 1982]; consideration is given only to when learnability is made possible or impossible, not to when it is made easy or difficult. This enables our results to be independent of particular learning algorithms and particular learnability criteria; however, it also restricts the analysis to representation shifts that are not information-preserving.

Proofs of theorems are included in an appendix.

2 Motivation and Overview

Consider a learning program with the architecture shown in figure 1. First, examples are mapped from an initial space X_I into the representation space X_R; then, a learning algorithm is run in the representation space; finally, the concept learned in the representation space is translated back to the initial space. Such a learning system is shifting the representation of examples from X_I to X_R using the function f_R.

Several learning systems of this sort have been built [Cohen, 1988; Cohen, 1990b; Flann and Dietterich,

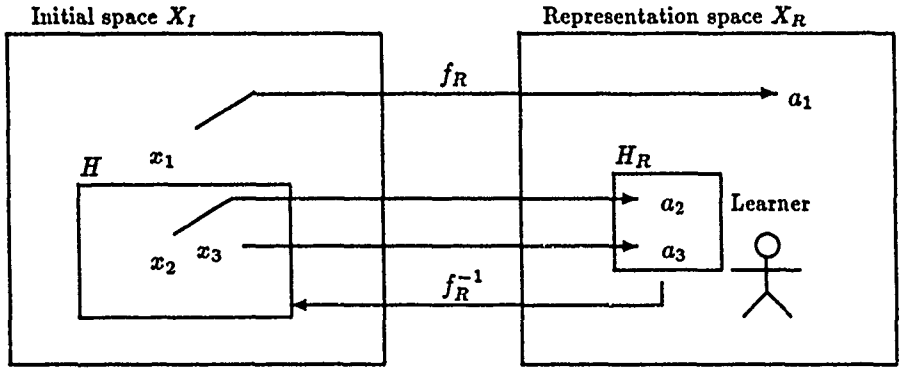


Figure 1: A learning system using representation shift

1989; Hirsh, 1989]. One advantage of this architecture is that it allows background knowledge to be applied to a learning problem in a highly modular way. Background knowledge is used only to *select the space over which learning will occur*, via the representation-shifting function f_R; standard concept learning techniques can then be applied in the new space X_R.

A small notational issue should be clarified at this point. In this paper, we will adopt the convention that if f is a function with domain X and if $S \subseteq X$, then $f(S)$ denotes the set $\{f(x) : x \in S\}$. This will greatly simplify the notation in situations where there are two parallel functions that must be considered, one which maps instances to instances, and one which maps concepts to concepts. An example of this appears in Figure 1, which actually contains two representation-shifting functions: f_R, which maps examples from the initial space to the representation space, and an "inverse" of f_R, denoted f_R^{-1} in the figure, which maps *concepts* in the representation space back to *concepts* in the initial space. The function f_R^{-1} is of course not a true inverse of f_R; rather, it it the inverse of f_R extended to concepts using the convention described above.

The effectiveness of such a learning system depends on the representation shift. The first issue addressed in this paper is: when is a representation shift appropriate for a learning problem? In short, what are the *principles* that underlie representation shift? We establish a correspondence called *knowledge-level equivalence* between a representation and the assumptions about the learning problem that are implicitly made when that representation is selected. This correspondence is the basis of our analytic framework.

The correspondence can be used in three ways.

- Given a representation shift, it is possible to identify the background knowledge to which a representation shift corresponds. In other words, it is possible to uncover the assumptions implicit in the use of a specific representation shift.
- Conversely, given some background knowledge about a learning problem, it is possible to test whether a given representation shift is necessarily appropriate, by testing whether the knowledge which corresponds to the representation shift is implied by the background knowledge.
- Finally, if one assumes that it is desirable to use a representation shift which corresponds to the strongest possible knowledge, it is possible to determine if a representation shift is optimal with respect to some given background knowledge about a learning system. This suggests a procedure for constructing a representation shift given some background knowledge about a learning problem. Such a representation shift is a means of incorporating background knowledge into a concept learning problem.

Two applications are given of our framework. In section 4.2, we find the assumptions implicit in the use of a common representational shift: a shift from examples to the "explanation structures" generated by the explanation based generalization (EBG) algorithm [Mitchell *et al.*, 1986]. This shift is found to correspond to an assumption about the correctness of EBG. In section 4.3, we relax this assumption of correctness, and construct a new representation shift which is knowledge-level equivalent to the relaxed assumption. This representation shift could be used as the basis of a learning system that makes a weaker assumption about the correctness of EBG.

3 A framework for analyzing representation shift

The question addressed in this section is: when is a representation shift appropriate to a particular class of learning problems? The meaning of this question is intuitively clear. However, in order to formulate and answer the question above precisely, it is necessary to formally define the terms involved: *representation shift*, *learning problem*, and *appropriate*.

3.1 What is a representation shift?

A *representation shift* will be formalized as a function f_R from an original *initial space* X_I into another space

X_R, called the *representation space.*

Representation shift $\equiv$ a function $f_R : X_I \rightarrow X_R$

In a typical case, X_I is a set of potential training examples in some initial description language, which is inappropriate for learning. X_R contains representations of the elements of X_I, in what is hopefully a better format for learning. For example X_I might be a set of digitized photographs of soybean plants, and X_R might consist of feature-vector representations of these plants using certain features known to be significant for the learning task at hand, as in [Michalski and Chilausky, 1980]; or X_I might be a set of chess positions, and X_R might be proofs that white has a forced exchange from that position, as in [Flann and Dietterich, 1989].

3.2 What is a learning problem?

A definition of a "class of learning problems" must now be given. It is assumed that the problem faced by the learner is to correctly identify some unknown *target concept* T given some "partial information" about T. Typical examples of the sorts of partial information available might be a set of answers to queries issued by the learner, or a set of labeled examples of members and nonmembers of T.

We would like to *exclude* from our formalization of a learning problem anything specific to a particular learning algorithm or a particular model of learnability. One way to do this is to assume that the partial information is requested by the learner (for example, via a set of calls to oracles) and is not an explicit input to the learner. (It seems reasonable to assume that for every learner that takes partial knowledge as an explicit input, there is a learner that requests this input from an oracle.) In this case, a *learning problem* is simply a possible target concept T, and a class of learning problems is simply a set of such concepts.

Class of learning problems $\equiv$ a set $\mathcal{P} = \{T_1, T_2, \ldots, \}$

3.3 When is a representation appropriate?

Given these definitions, it is now possible to state precisely when a representation is appropriate. *A representation f_R is appropriate for the class of learning problems $\mathcal{P}$ if and only if each learning problem is potentially solvable in the representation space.* A learning problem T is *potentially solvable* if it is possible to exactly describe a target concept T by an image of T under f_R. If we let $APPROP(f_R, \mathcal{P})$ denote the proposition "f_R is appropriate for $\mathcal{P}$", and let X_R be the range of f_R, then the definition of appropriateness can be stated in symbols:

$$APPROP(f_R, \mathcal{P}) \equiv$$
$$f_R \text{ is a well-defined function} \wedge$$
$$\forall T \in \mathcal{P},\ \exists T_R \subseteq X_R : T = f_R^{-1}(T_R)$$

An alternative definition which some readers may find more intuitive is

$$APPROP(f_R, \mathcal{P}) \equiv$$
$$f_R \text{ is a well-defined function} \wedge$$
$$\forall T \in \mathcal{P},\ \forall x, y \in X_I,$$
$$(f_R(x) = f_R(y) \Rightarrow \mathcal{X}_T(x) = \mathcal{X}_T(y))$$

where $\mathcal{X}_T(x)$ denotes the characteristic function of T re.

e definition of potentially solvable assumes that the goal of learning is exact identification of the target concept. This fits many formal definitions of learnability, such as Gold's learnability in the limit [Gold, 1967], Littlestone's mistake-bounded learnability [Littlestone, 1988], and Angluin's definitions of learning from queries [Angluin, 1988]. An exception is the Valiant criterion of probably approximately correct (pac) learnability [Valiant, 1984], which requires only approximate identification of the target concept; this suggests that a probabilistic extension of this analytic framework may be worth investigating.

Notice that the appropriateness of f_R does not say anything about how easy or difficult it is to solve the learning problem in the representation space X_R; it merely says that it is still *possible* to solve the learning problem.

3.4 Representations and background knowledge

Let $\mathcal{A}(\mathcal{P})$ denote a first order sentence which is a statement about $\mathcal{P}$; $\mathcal{A}(\mathcal{P})$ will alternatively be viewed as an assumption about the class of learning problems, or as background knowledge about the class of learning problems. We can now formulate precisely and answer the questions given in the introduction:

Question 1: Given a representation f_R what assumptions are implicitly made when f_R is used on the class of learning problems $\mathcal{P}$?

The answer to this question is obvious: *any learning system that uses a representation f_R makes the assumption that f_R is appropriate for the learning problem; i.e.,* that $APPROP(f_R, \mathcal{P})$ is true. A particular learner may (and probably will) make other assumptions as well in the process of generating a hypothesis; however, since *every* learning system that uses the representation f_R assumes $APPROP(f_R, \mathcal{P})$, then it seems reasonable to characterize $APPROP(f_R, \mathcal{P})$ as the assumptions made in choosing the representation f_R.

Question 2a: Given some background knowledge $\mathcal{A}(\mathcal{P})$ about a class of learning problems $\mathcal{P}$, what representations f_R can be used?

Question 2b: In the same circumstances, what representations f_R *should* be used?

The answer to question 2a is also obvious: if

$$\mathcal{A}(\mathcal{P}) \Rightarrow APPROP(f_R, \mathcal{P}) \qquad (1)$$

then the representation f_R *could* be used for the learning problem $\mathcal{P}$, given the background knowledge $\mathcal{A}(\mathcal{P})$. The answer to 2b is not obvious, however. There may be many representations that have this property; which of these *should* be used? In short, are there any grounds for preferring one of these representations over others?

One context in which f_{R_1} is preferable to f_{R_2} is when the appropriateness of f_{R_1} *implies* the appropriateness of f_{R_2}, *i.e.*

$$APPROP(f_{R_1}, \mathcal{P}) \Rightarrow APPROP(f_{R_2}, \mathcal{P}) \quad (2)$$

The rational for preferring f_{R_1} is the following: consider a learning system L_1 that uses f_{R_1}, and another learning system L_2 that uses f_{R_2}. L_1 "knows" only that f_{R_1} is appropriate, and L_2 "knows" only that f_{R_2} is appropriate. Then if equation 2 holds, *everything known by L_2 is deductively implied by what is known by L_1*, and can, at least in principle, be derived from knowledge available to L_1. At the knowledge level, L_1 is "better informed about the learning problem" than L_2.

This rule of preference means that logical implication imposes a partial order on the desirability of representations. The best representations are those f_R that satisfy equation 1 and are maximal with respect to this partial order. It is easy to show that if f_R has the property

$$\mathcal{A}(\mathcal{P}) \iff APPROP(f_R, \mathcal{P})$$

then it fulfills this requirement. Such a representation will said to be *knowledge-level equivalent* (KL-equivalent) to the assumption $\mathcal{A}(\mathcal{P})$. The answer to question 2b is thus: *given $\mathcal{A}(\mathcal{P})$, the most desirable representation is some f_R such that f_R is KL-equivalent to $\mathcal{A}(\mathcal{P})$.*

Notice that f_R is not unique: that is, given an assumption $\mathcal{A}(\mathcal{P})$, there is more than one f_R that is KL-equivalent to $\mathcal{A}(\mathcal{P})$. In particular, if g is any one-to-one function and f_R is KL-equivalent to $\mathcal{A}(\mathcal{P})$, then $f_R \circ g$ is also KL-equivalent to $\mathcal{A}(\mathcal{P})$. This is to be expected: if g is one-to-one, then it is an "information-preserving" operation and should make no difference at the knowledge level. However, the following theorem shows that this is the only sort of variation of f_R which is possible: *i.e.*, that the KL-equivalent representation for an assumption $\mathcal{A}(\mathcal{P})$ is unique up to composition with one-to-one functions.

Theorem 1 *If f_{R_1} and f_{R_2} are both KL-equivalent to $\mathcal{A}(\mathcal{P})$, then there is a one-to-one function g such that $f_{R_1} = f_{R_2} \circ g$.*

The analytic framework presented in this section is summarized in table 1.

4 Applications of the framework

The true value of an analytic framework lies in its usefulness in the analysis of meaningful problems. We will consider two uses of the framework. First, in section 4.2, we find the assumption implicit in the use of a common representational shift. This is a simple analysis: it merely requires finding a meaningful condition which is logically equivalent to $APPROP(f_R, \mathcal{P})$. In section 4.3, we consider the related problem of constructing a representation shift which is optimal given a particular assumption about a learning problem. Here, given an assumption about the learning problem, the problem is to construct a representation shift which is KL-equivalent to this assumption.

The representation shifts and assumptions which we consider are closely connected to the operation of explanation based generalization (EBG), a procedure that uses a logical theory to generalize a single positive example of some concept. A short discussion of EBG precedes our analyses.

4.1 Explanation based generalization

To avoid unnecessary detail, explanation based generalization will be discussed at a rather abstract level in this paper. Readers requiring more detail are referred to [Kedar-Cabelli and McCarty, 1987; Mitchell *et al.*, 1986]. Let Θ be a theory, and for $x \in X_I$, let $PROOFS_\Theta(x)$ denote the set of proofs of x in Θ. This assumes that x is a logical goal, for example *cup(obj1)*; such an example is implicitly tagged with the "target concept" to which it is relevant. Let $\mathcal{O}$ denote an operationality predicate; in this context, an operationality predicate tells if a subproof is a "detail" not relevant to the concept to be learned. *Explanation based generalization* is an operation defined on a proof $\gamma_x \in PROOFS_\Theta(x)$ and denoted by the function $EBG(\Theta, \mathcal{O}, p_x)$. This function always returns a set that generalizes x; *i.e.*, $\forall x \in X_I$, $\{x\} \subseteq EBG(\Theta, \mathcal{O}, p_x) \subseteq X_I$.

Notice that the usual assumptions about Θ — *e.g.*, that it is complete and correct, that it is defined only for positive examples — have not been made. The reason is that the assumptions which an EBL system makes about Θ depend in part on *how generalizations are used*. The remainder of this section will consider several possible schemes for using EBG, and analytically determine which assumptions about the domain theory are being made.

There are several known algorithms for explanation based generalization. Generally, a proof p_x is generalized in two ways: operational subtrees are pruned in accordance with the $\mathcal{O}$ predicate, and some of the constants appearing in the proof are replaced with variables. This generalized proof is called an *explanation structure*, and will be denoted $e_\mathcal{O}(p_x)$. A rule is then extracted from the explanation structure by conjoining the leaves of the explanation structure to form the antecedent, and using the root of the explanation structure as the consequent. The set of goals provable directly from this rule is the generalization of x.

As an example of explanation based generalization, [DeJong and Mooney, 1986] describes a simple theory of social interactions, under which the goal *kill(john,john)* is generalized to form the rule

$$\text{kill(X, X)} \leftarrow \text{depressed(X)} \wedge \text{buy(X, W)} \wedge \text{gun(W)}$$

which can be paraphrased as saying "*X* will kill himself if he is depressed and has bought a gun." Here the example is the goal *kill(john,john)* and the generalization is the set of goals provable by the rule above, *i.e.* the set of goals of the form *kill(X,X)* such that

$$\text{depressed(X)} \wedge \text{buy(X, W)} \wedge \text{gun(W)}$$

is provable. In this case, the generalization corresponds loosely to the concept of "suicide".

Table 1: Summary of the analytic framework

Intuitive Notion	Formalization
representation shift	a function $f_R : X_I \rightarrow X_R$
learning problem	a target concept $T \subseteq X_I$
class of learning problems	a set $\mathcal{P} = \{T_1, T_2, \ldots\}$
background knowledge of learning problem	a first order formula $\mathcal{A}(\mathcal{P})$
f_R is appropriate for $\mathcal{P}$	$APPROP(f_R, \mathcal{P})$
assumption implicitly made in using f_R	$APPROP(f_R, \mathcal{P})$
f_R can be used given $\mathcal{A}(\mathcal{P})$	$\mathcal{A}(\mathcal{P}) \Rightarrow APPROP(f_R, \mathcal{P})$
f_R should be used given $\mathcal{A}(\mathcal{P})$	$\mathcal{A}(\mathcal{P}) \Longleftrightarrow APPROP(f_R, \mathcal{P})$
	(*i.e.*, f_R is KL-equivalent to $\mathcal{A}(\mathcal{P})$)

Later analysis will not depend on the precise algorithm used for EBG; however, the following property is assumed to hold.[1]

Assumption 1 *For all* Θ, $\mathcal{O}$, p_x, *and* y,

$$y \in EBG(\Theta, \mathcal{O}, p_x) \iff \exists p_y \in PROOFS_\Theta(y) : e_\mathcal{O}(p_y) = e_\mathcal{O}(p_x)$$

4.2 Analysis of a representation shift

An important topic in machine learning research is integration of explanation based and similarity based learning techniques. Several integrated learning systems have been built that have the architecture of figure 1, and that use as a representation-shifting function some close variant of the function $f(x) \equiv e_\mathcal{O}(p_x)$ These systems *learn from a set of explanation structures* using SBL techniques; they differ primarily in the learning methods used. (See [Flann and Dietterich, 1989] for a discussion of some of these systems.)

What assumptions are made by a learning program that that uses this representation shift? The answer $APPROP(f, \mathcal{P})$ is correct, but not very meaningful. However, it can be shown that $APPROP(f, \mathcal{P})$ is logically equivalent to a meaningful assumption about the behavior of EBG.

Theorem 2 *Let* $f_{\Theta,\mathcal{O}}$ *be defined as* $f_{\Theta,\mathcal{O}}(x) \equiv e_\mathcal{O}(p_x)$. *Then*

$$APPROP(f_{\Theta,\mathcal{O}}, \mathcal{P}) \equiv$$
$$\forall x \in X_I, |e_\mathcal{O}(PROOFS_\Theta(x))| = 1 \ \wedge$$
$$\forall T \in \mathcal{P}, \forall x \in T, \forall p_x \in PROOFS_\Theta(x),$$
$$EBG(\Theta, \mathcal{O}, p_x) \subseteq T$$

[1]This assumption should hold for any reasonable implementation of EBG. The "only if" direction requires only that the output of EBG be determined by Θ,$\mathcal{O}$, and $e_\mathcal{O}(p_x)$: in this case, if $\exists p_y \in PROOFS_\Theta(y) : e_\mathcal{O}(p_y) = e_\mathcal{O}(p_x)$, then $EBG(\Theta, \mathcal{O}, p_y) = EBG(\Theta, \mathcal{O}, p_x)$. We know that $y \in EBG(\Theta, \mathcal{O}, p_y)$, and hence $y \in EBG(\Theta, \mathcal{O}, p_x)$ as well. The "if" direction is also intuitively clear, if the conjunction that describes the set $EBG(\Theta, \mathcal{O}, p_x)$ represents preconditions under which the "abstract version" $e_\mathcal{O}(p_x)$ of the proof p_x is applicable: if y is in this set, those preconditions are true, and some instantiation of the "abstract proof" $e_\mathcal{O}(p_x)$ should succeed for y as well as for x.

This restatement of $APPROP$ shows that it is equivalent to two assumptions about EBG. The first assumption is that every instance has at most a single proof, up to the level of detail specified by the explanation structure. Since this constraint is usually enforced by ensuring that there is at most one proof for every instance in the domain theory, we will call this assumption the *unique proof assumption*. The second assumption is about the "validity" of generalizations of positive examples. Intuitively, the generalization of any $x \in T$ is *valid*: *i.e.*, it is a correct generalization of x with respect to T. Notice that this assumption is much weaker than the assumption, commonly associated with EBL systems, that the theory is a complete and correct definition of the target concept T.

Theorem 2 has the interesting corollary that f is KL-equivalent to the assumption above (*i.e.*, the conjunction of the unique proof assumption and the validity assumption.) This means that if L' is *any* hybrid SBL/EBL concept learning system which makes these assumptions, and which assumes nothing else about the learning problem, then a learner that uses the representation shift $f_{\Theta,\mathcal{O}}$ will have access to exactly the same knowledge that is available to L'. This is true regardless of the architecture of L'.

4.3 Constructing a representation shift

Given the analysis above, it is natural to ask: in what ways can these assumptions about EBG be relaxed? One class of theory that violates these assumptions and that appears to occur often in practice is the class of "abductive" theories: theories that make assumptions in the process of theorem proving. These theories often violate the unique proof assumption. A common subclass of abductive theories is the class of theories that involve plan recognition; often, any of several assumptions could be introduced that would suffice to explain an action, but only explanations based on the right assumption will be correct.

The crucial property of an abductive theory seems to be that *for every true observation, there is some correct explanation.* It seems reasonable to require that a generalization associated with a valid explanation is valid. This "abductive assumption" about the correct-

ness of EBG can be precisely stated as:

$$\mathcal{AB}_{\Theta,\mathcal{O}}(\mathcal{P}) \equiv$$
$$\forall T \in \mathcal{P}, \forall x \in T,$$
$$\exists p_x \in PROOFS_\Theta(x) : EBG(\Theta, \mathcal{O}, p_x) \subseteq T$$

A reasonable application of our framework would be to find a representation shift which is optimal under this assumption: that is, a representation shift which is KL-equivalent to $\mathcal{AB}_{\Theta,\mathcal{O}}(\mathcal{P})$. A learning algorithm using this representation would have the advantages of systems which use the representation shifting function f, but would be useful even when the unique proof assumption does not hold.

In order to find a representation that is KL-equivalent to this assumption, it appears necessary to make some assumptions about $\mathcal{P}$ as well. Let us say that a set $S \subseteq 2^X$ is a *hitset* iff $\exists S_0 \subseteq X : S = \{R : \exists r \in R \cap S_0\}$. A class of learning problems $\mathcal{P}$ will be said to have *hitset images under* f_R iff $\forall T \in \mathcal{P}$, $f_R(T)$ is a hitset.

Theorem 3 *Let the function* $f^*_{\Theta,\mathcal{O}}(x)$ *be defined as* $f^*_{\Theta,\mathcal{O}}(x) \equiv \{e_\mathcal{O}(p_x) : p_x \in PROOFS_\Theta(x)\}$. *If* $\mathcal{P}$ *has hitset images*[2] *under* $f^*_{\Theta,\mathcal{O}}$, *then* $f^*_{\Theta,\mathcal{O}}$ *is KL-equivalent to* $\mathcal{AB}_{\Theta,\mathcal{O}}(\mathcal{P})$.

Discussion of learning algorithms for the representation space which is the range of f^* is outside the scope of this paper; however, the A-EBL technique described in [Cohen, 1989; Cohen, 1990a; Cohen, 1990b] can be viewed such a learning algorithm.

A few remarks are in order about the theorem above, in particular about the requirement that $\mathcal{P}$ have "hitset images". First, this property generally holds if $\mathcal{P}$ contains only concepts that can be expressed using a set of rules produced by EBG; in particular, it can easily shown that if assumption 1 holds, then the image under f^* of the set $EBG(\Theta, \mathcal{O}, p_x)$ is always a hitset.

Second, the proof of the theorem shows that the property of having hitset images is *not* necessary to establish that

$$\mathcal{AB}_{\Theta,\mathcal{O}}(\mathcal{P}) \Rightarrow APPROP(f^*_{\Theta,\mathcal{O}}, \mathcal{P})$$

The property is only necessary for the other side of the logical equivalence which is the definition of KL-equivalence; that is, having hitset images is only necessary to show that

$$APPROP(f^*_{\Theta,\mathcal{O}}, \mathcal{P}) \Rightarrow \mathcal{AB}_{\Theta,\mathcal{O}}(\mathcal{P})$$

In other words, the function f^* will be an appropriate representation shift *whenever* $\mathcal{AB}(\mathcal{P})$ holds; the condition that $\mathcal{P}$ has hitset images is only needed to show that f^* is the best representation shift.

[2]Of course, the assumption that $\mathcal{P}$ has hitset images could be simply added to $\mathcal{AB}(\mathcal{P})$. This presentation emphasizes the fact that this assumption is motivated by the desire to construct a KL-equivalent representation shift rather than by a natural assumption about the learning problem.

4.4 Discussion of the results

Any system which uses EBG must make an assumption about the domain theory which is used; the assumption that is commonly associated with EBG is that the domain theory is a complete, correct, and tractable definition of the target concept. In recent years, several hybrid learning systems have been built that have the architecture of figure 1, and which use as a representation-shifting function a mapping from instances to the explanation structures associated with those instances. Presumably, these systems make a weaker assumption about the correctness of the domain theory.

The analysis above confirms this intuition. It shows that the domain theory used by such a learning system must satisfy two conditions: every instance must have exactly one associated explanation structure, and EBG must never overgeneralize a positive example. This is a strong assumption, but it does *not* require the domain theory to be complete and correct; in particular, the domain theory might produce an (incorrect) proof for a negative example of the concept. In other words, the domain theory can be overly general with respect to the target concept. However, the theory must be sufficiently detailed that EBG does not overgeneralize any positive examples.

The second result shows that some of the requirements on the domain theory can be relaxed by using a slightly different abstraction function. By mapping an instance to a *set* of explanation structures, rather than to a single explanation structure, a somewhat weaker requirement is placed on the domain theory. The domain theory must still be complete; however, the theory may produce multiple inconsistent explanation structures for each instance, rather than a single explanation structure. The theory must still be sufficiently detailed that EBG does not overgeneralize on *all* explanations of a positive example; however, EBG may overgeneralize on *some* of the possible explanations of an instance.

The second result is also of interest because it shows that it is possible to relax assumptions about the domain theory in a principled way, and thus to expand the range of competence of a representation-shifting learning system.

5 Conclusions

In this paper, we have developed a framework for knowledge-level analysis of the changes in representations of training examples in learning. There have been several learning systems implemented that use changes of representation of the kind considered in this paper; some good examples are [Cohen, 1990b] and the systems discussed in [Flann and Dietterich, 1989].[3] The

[3]It is important to keep in mind the difference between these systems and work on constructive induction, for example [Drastal *et al.*, 1989]: constructive induction is concerned with *information-preserving* transformations of training instances.

framework was used to uncover the assumptions implicit in a commonly-used representation shift, and to derive a representation shift which is appropriate for a weak assumption about the correctness of EBL. The first result is important because it helps us to understand the assumptions made by the learning systems which use this representation shift; the second, because it shows how to relax these assumptions in a principled way.

Most prior analyses of representation change have focused on automatic reformulation of problem-solving strategies [Amarel, 1981; Korf, 1981; Riddle, 1989; Subramanian, 1989]. Some of the formal models of representation change in problem solving, however, are general enough to serve as a model for representation change in learning. In particular, our model of representation change in learning and notion of appropriateness is closely related to the models presented in [Holte and Zimmer, 1989] and [Lowry, 1988]; in fact, it is a special case, in which the problem to be solved is required to be a learning problem. However, the earlier models do not develop the notion of a "most appropriate" representation, and have not been applied to analysis of representation shifts in learning.

Work in representation change in learning has generally focused on automatic change or derivation of inductive bias [Chrisman, 1989; Keller, 1989; Russello and Grosof, 1989; Utgoff, 1984]. The representation changes considered in this paper – changes in the representation of training examples – are more fundamental than shift of bias; such a change alters the very space over which learning occurs, and hence necessitates selection of a new hypothesis space and (probably) a new learning algorithm. Our goals, however, are more modest: the goal of this paper is to provide a framework for analysis of representational choices made by humans, and to suggest a methodology for humans to follow in constructing new representational shifts, given background knowledge about a learning problem.

These goals have been at least partially met, however, many problems remain for future research. One difficult problem is automation (or partial automation) of the methodology suggested for constructing representational shifts. Another set of problems is suggested by the observation that there is a second possible interpretation of a the architecture of figure 1: X_I might be the space in which training examples are "naturally found" (for instance, X_I might be the set of all soybean plants). In this case, f_R is a noncomputable function that denotes an *initial choice* of representations. Our basic framework applies to initial-choice representations as well as to representation-shifting functions, although obviously proof techniques for showing the appropriateness of initial-choice representations will be different from the proof techniques used in this paper. Finally, the definition of the appropriateness of a representational shift assumes that the goal of learning is exact identification of the target concept; this is a more stringent identification criterion than the Valiant criterion of probably approximately correct learnability [Valiant, 1984], which requires only approximate identification of the target concept. This suggests that a probabilistic extension of the analytic framework may be worth investigating.

6 Acknowledgements

I am grateful to Alex Borgida, for his advice and encouragement; Chun Liew, Patricia Riddle, and Haym Hirsh, for valuable comments on a draft of the paper; and to many other members of the Rutgers AI community. The paper was also much improved by the comments of an anonymous reviewer (Rob Holte.) The author is supported by an AT&T Fellowship. Initial stages of this research were supported by a Marion Johnson Fellowship.

A Proofs of theorems

A.1 Proof of theorem 1

If $\mathcal{X}_T(x)$ is the characteristic function of T on x then it can be easily verified that

$$APPROP(f_A, \mathcal{P}) \equiv \tag{3}$$

$$\forall T \in \mathcal{P},\ \forall x, y \in X_I,\ (f_R(x) = f_R(y) \Rightarrow \mathcal{X}_T(x) = \mathcal{X}_T(y)) \tag{4}$$

So if

$$\begin{aligned} & \mathcal{A}(\mathcal{P}) \\ \iff\ & APPROP(f_{R_1}, \mathcal{P}) \\ \iff\ & APPROP(f_{R_1}, \mathcal{P}) \end{aligned}$$

then for $i = 1, 2$

$$\forall x, y \in X_I,\ f_{R_i}(x) = f_{R_i}(y) \iff (\forall T \in \mathcal{P} \mathcal{X}_T(x) = \mathcal{X}_T(y))$$

and hence

$$f_{R_1}(x) = f_{R_1}(y) \iff f_{R_2}(x) = f_{R_2}(y)$$

and the mapping $g(x) = f_{R_2}(f_{R_1}^{-1}(x))$ is well-defined and one-to-one. ■

A.2 Proof of theorem 2

To prove the theorem, it is necessary to show both sides of the logical equivalence. The proposed simplification of $APPROP(f_{R1}, \mathcal{P})$ will be referred to as $\mathcal{A}(\mathcal{P})$.

To show that ($f_{\Theta,\mathcal{O}}$ is appropriate for $\mathcal{P}$) $\Rightarrow$ $\mathcal{A}(\mathcal{P})$. First, note that if $f_{\Theta,\mathcal{O}}$ is a well-defined function over X_I, then the unique proof assumption holds. If $f_{\Theta,\mathcal{O}}$ is also appropriate for $\mathcal{P}$, then

$$\forall T \in \mathcal{P},\ \forall x, y,\ \neg(f_{\Theta,\mathcal{O}}(x) = f_{\Theta,\mathcal{O}}(y) \wedge \mathcal{X}_T(x) \neq \mathcal{X}_T(y))$$

again using $\mathcal{X}_T(x)$ to denote the characteristic function of T on x. Now, if $f_{\Theta,\mathcal{O}}(x) = f_{\Theta,\mathcal{O}}(y)$ then there is some explanation structure e such that $f_{\Theta,\mathcal{O}}(x) = f_{\Theta,\mathcal{O}}(y) = e$, and

$$\begin{aligned} & \forall T \in \mathcal{P}, \forall x \in T,\ (e_{\mathcal{O}}(p_x) = e_{\mathcal{O}}(p_y) \Rightarrow y \in T) \\ \Rightarrow\ & \forall T \in \mathcal{P}, \forall x \in T,\ (y \in EBG(\Theta, \mathcal{O}.p_x) \Rightarrow y \in T) \\ \Rightarrow\ & \forall T \in \mathcal{P}, \forall x \in T,\ EBG(\Theta, \mathcal{O}, p_x) \subseteq T \end{aligned} \tag{5}$$

where p_x is any proof of x, and where p_y is any proof of y; hence the second conjunct holds. Line 5 follows from assumption 1.

To show that $\mathcal{A}(\mathcal{P}) \Rightarrow$ *($f_{\Theta,\mathcal{O}}$ is appropriate for $\mathcal{P}$).* First, note that the unique proof assumption implies that $f_{\Theta,\mathcal{O}}$ is a function. Second, observe that for any $\mathcal{P}$,

$$\begin{aligned} & \forall T \in \mathcal{P},\ \forall z \in T,\ (EBG(\Theta,\mathcal{O},p_x) \subseteq T) \\ \Rightarrow\ & \forall T \in \mathcal{P},\ \forall x \in \overline{T},\ (EBG(\Theta,\mathcal{O},p_x) \subseteq \overline{T}) \end{aligned}$$

where p_x again denotes a proof of x. To see that this is true, imagine that the antecedent holds but

$$\exists T \in \mathcal{P}, x \in \overline{T}, p \in PROOFS_\Theta(x) : EBG(\Theta,\mathcal{O},p_x) \not\subseteq \overline{T}$$

Then there is some $y \in EBG(\Theta,\mathcal{O},p_x) - \overline{T}$; note that this implies that y is in T. But by assumption 1

$$\begin{aligned} & y \in EBG(\Theta,\mathcal{O},p_x) \\ \Rightarrow\ & e_{\mathcal{O}}(p_y) = e_{\mathcal{O}}(p_x) \\ \Rightarrow\ & EBG(\Theta,\mathcal{O},p_y) = EBG(\Theta,\mathcal{O},p_x) \end{aligned}$$

and so

$$\exists T \in \mathcal{P}, y \in T : EBG(\Theta,\mathcal{O},p_y) \not\subseteq T$$

a contradiction. The preceding argument means that $\mathcal{A}(\mathcal{P})$ implies

$$\forall T \in \mathcal{P},\ \forall x,y \in X_I,\ (y \in EBG(\Theta,\mathcal{O},p_x) \Rightarrow \mathcal{X}_T(x) = \mathcal{X}_T(y))$$

Since by assumption 1

$$\begin{aligned} & y \in EBG(\ \ ),\mathcal{O},x) \\ \Longleftrightarrow\ & e_{\mathcal{O}}(p_y)\ = e_{\mathcal{O}}(p_x) \\ \Longleftrightarrow\ & f_{\Theta,\mathcal{O}}(x) = f_{\Theta,\mathcal{O}}(y) \end{aligned}$$

it follows that

$$\forall T \in \mathcal{P},\ \forall x,y,\ (f_{\Theta,\mathcal{O}}(x) = f_{\Theta,\mathcal{O}}(y) \Rightarrow \mathcal{X}_T(x) = \mathcal{X}_T(y))$$

Hence $f_{\Theta,\mathcal{O}}$ is appropriate by equation 3. ■

A.3 Proof of theorem 3

Again, both sides of the equivalence must be shown.

To show $\mathcal{AB}(\mathcal{P}) \Rightarrow f^*_{\Theta,\mathcal{O}}$ *is appropriate.* The contrapositive will be shown. If $f^*_{\Theta,\mathcal{O}}$ is not appropriate, then

$$\exists T \in \mathcal{P}, x,y : f^*_{\Theta,\mathcal{O}}(x) = f^*_{\Theta,\mathcal{O}}(y) \wedge \mathcal{X}_T(x) \neq \mathcal{X}_T(y)$$

Assume without loss of generality that $x \in T$ and $y \in \overline{T}$. Now, by assumption 1, if $f^*_{\Theta,\mathcal{O}}(x) = f^*_{\Theta,\mathcal{O}}(y)$ then $\forall p_x \in PROOFS(x), y \in EBG(\Theta,\mathcal{O},p_x)$. But then

$$\exists T \in \mathcal{P}, x \in T, y \in \overline{T} : \forall p_x \in PROOFS(x),\ y \in EBG(\Theta,\mathcal{O},p_x)$$

contradicting $\mathcal{AB}(\mathcal{P})$.

To show $f^*_{\Theta,\mathcal{O}}$ *is appropriate* $\Rightarrow \mathcal{AB}(\mathcal{P})$. This direction of the proof uses the easily-verified proposition that for any appropriate representation f_R for $\mathcal{P}$

$$\forall T \in \mathcal{P},\ \forall x \in X_I,\ (x \in T \Longleftrightarrow f_R(x) \in f_R(T))$$

If $f^*_{\Theta,\mathcal{O}}$ is appropriate for $\mathcal{P}$ then

$$\forall T \in \mathcal{P}, \forall x \in X_I,\ (x \in T \Rightarrow f^*_{\Theta,\mathcal{O}}(x) \in f^*_{\Theta,\mathcal{O}}(T))$$

Now, since $f^*_{\Theta,\mathcal{O}}(T)$ is a hitset, there is some E_0 such that $f^*_{\Theta,\mathcal{O}}(T) = \{E : \exists e_{\mathcal{O}}(p_{xi}) \in E \cap E_0\}$ and so $f^*_{\Theta,\mathcal{O}}(x) \in f^*_{\Theta,\mathcal{O}}(T) \Rightarrow \exists e_{\mathcal{O}}(p_{xi}) \in f^*_{\Theta,\mathcal{O}}(x) \cap E_0$

Let e' denote the explanation structure in $f^*_{\Theta,\mathcal{O}}(x) \cap E_0$. We claim that any y that has a proof p_y with an explanation structure equivalent to e' will be in T. If this is true, then by assumption 1, $EBG(\Theta,\mathcal{O},p_x) \subseteq T$ for the proof $p_{xi'}$ of x with explanation structure e'; and since nothing has been assumed except that $x \in T$ and $T \in \mathcal{P}$, $\mathcal{AB}(\mathcal{P})$ will be established and the proof will be complete.

But the claim must be true, because for any y,

$$\begin{aligned} & e' \in f^*_{\Theta,\mathcal{O}}(y) \\ \Rightarrow\ & e' \in f^*_{\Theta,\mathcal{O}}(y) \cap E_0 \\ \Rightarrow\ & f^*_{\Theta,\mathcal{O}}(y) \in f^*_{\Theta,\mathcal{O}}(T) \\ \Rightarrow\ & y \in T \end{aligned}$$

the last step following because $f^*_{\Theta,\mathcal{O}}$ is appropriate. ■

References

[Amarel, 1981] Saul Amarel. On representations of problems of readings about actions. In *Readings in Artificial Intelligence.* Morgan Kaufmann, 1981.

[Angluin, 1988] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4), 1988.

[Chrisman, 1989] Lonnie Chrisman. Evaluating bias during pac-learning. In *Proceedings of the Sixth International Workshop on Machine Learning.* Morgan Kaufmann, 1989.

[Cohen, 1988] William W. Cohen. Generalizing number and learning from multiple examples in explanation-based learning. In *Proceedings of the Fifth International Conference on Machine Learning.* Morgan Kaufmann, 1988.

[Cohen, 1989] William W. Cohen. Abductive explanation based learning: A solution to the multiple explanation problem. Technical Report ML-TR-26, Rutgers University, 1989.

[Cohen, 1990a] William W. Cohen. An analysis of representation shift in concept learning. In *Proceedings of the Seventh International Conference on Machine Learning.* Morgan Kaufmann, 1990.

[Cohen, 1990b] William W. Cohen. Learning from textbook knowledge. In *Proceedings of the Eighth National Conference on Artificial Intelligence.* AAAI, 1990.

[DeJong and Mooney, 1986] Gerald DeJong and Raymond Mooney. EBL: An alternative view. *Machine Learning*, 1(2), 1986.

[Drastal *et al.*, 1989] George Drastal, Gabor Czako, and Stan Raatz. Induction in abstraction spaces: A form of constructive induction. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* IJCAI, 1989.

[Flann and Dietterich, 1989] Nicholas Flann and Thomas Dietterich. A study of explanation-based methods for inductive learning. *Machine Learning*, 4(2), 1989.

[Gold, 1967] Mark Gold. Language identification in the limit. *Information and Control*, 10, 1967.

[Hirsh, 1989] Haym Hirsh. Incremental version space merging: A general framework for concept learning. PhD Thesis, Stanford University Department of Computer Science, 1989.

[Holte and Zimmer, 1989] Robert Holte and Robert Zimmer. A mathematical framework for studying representations. In *Proceedings of the Sixth International Workshop on Machine Learning*. Morgan Kaufmann, 1989.

[Kedar-Cabelli and McCarty, 1987] Smadar Kedar-Cabelli and L. Thorne McCarty. Explanation-based generalization as resolution theorem proving. In *Proceedings of the Fourth International Workshop on Machine Learning*. Morgan Kaufmann, 1987.

[Keller, 1989] Rich Keller. Compiling learning vocabulary from perfcrmance description. In *Proceedings of the Sixth International Workshop on Machine Learning*. Morgan Kaufmann, 1989.

[Korf, 1981] Richard Korf. Toward a model of representation change. *Artificial Intelligence*, 14:41–78, 1981.

[Littlestone, 1988] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4), 1988.

[Lowry, 1988] Michael Lowry. STRATA: Problem reformulation and abstract data types. In *Change of Representation and Inductive Bias*. Kluwer Academic Publishers, 1988.

[Michalski and Chilausky, 1980] R.S. Michalski and R. L. Chilausky. Learning from being told and learning from examples: an experimental comparision of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *Policy Analysis and Information Systems*, 4, 1980.

[Mitchell *et al.*, 1986] T. Mitchell, R. Keller, and S. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1), 1986.

[Newell, 1982] Allen Newell. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982.

[Riddle, 1989] Patricia Riddle. Reformulation from state space to reduction space. In *Proceedings of the Sixth International Workshop on Machine Learning*. Morgan Kaufmann, 1989.

[Russello and Grosof, 1989] Stuart Russello and Benjamin Grosof. Declarative bias for structural domains. In *Proceedings of the Sixth International Workshop on Machine Learning*. Morgan Kaufmann, 1989.

[Subramanian, 1989] Devika Subramanian. Representational issues in machine learning. In *Proceedings of the Sixth International Workshop on Machine Learning*. Morgan Kaufmann, 1989.

[Utgoff, 1984] Paul Utgoff. Shift of bias for inductive concept learning. Technical Report CBM-TR-145, Rutgers University, 1984.

[Valiant, 1984] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11), November 1984.

Learning Procedures by Environment-Driven Constructive Induction

David V. Hume
Department of Computer Science,
University of New South Wales,
P.O. Box 1, Kensington NSW, Australia 2033.

Abstract

Induction of action sequences in a simulated robot world is described. The learner passively observes examples which are procedural action sequences where properties and/or relations change in the simulation. The observed procedures are split and trimmed to form the initial concept descriptions. Active experimentation occurs when the system repeats or tests more general descriptions of the observed examples. Generalisations are formed by constructive induction using inverse resolution and tested by execution of operational actions in the simulation. Completed execution of the test confirms success. Inability to complete a concept in the simulation causes search for successively greater generalisation on demand inorder to continue execution. Background knowledge need not be present as the system invents appropriately justified concepts as required.

1 Introduction

Before children learn conventional language they can learn how to build columns and arches from blocks, and other such concepts. They do this by observing an agent (maybe a parent) perform a one or more examples of these concepts, and then imitating the observed actions. Once they start experimenting they can sometimes proceed and successfully learn the intended concept, or some variation on it, without feedback from the agent ("playing by themselves").

In a simulated world containing two robots we examine this type of learning from the point of view of the child. Thus the learning task is. *Given* an intelligent agent performing examples of purposeful action sequences, *Create* a learning element which observes these examples and then conducts experiments to acquire useful concepts by exploring the environment.

The learning of procedures from examples in a reactive environment has received some attention in. NODDY {Andreae, 1985} (learns blocks world actions), SIERRA {VanLehn, 1987} (learns how to subtract) and BAGGER2 {Shavlik, 1989} (learns wide variety of concepts: build column, transform logic circuit, etc.). However, each acquires concepts by passively observing examples without generating experiments. Active learning of single step procedures occurs cooperatively in ADEPT (experimentation) and PIIINEAS (concept revision) {Falkenhainer & Rajamoney, 1988} with explanations of new physical processes. But in all these systems, as for EBL in general, extensive domain theory must be supplied or built in so that explanations of examples may be analysed.

In planning or problem solving, two learning systems which can learn temporal concepts interactively and autonomously in an environment are PRODIGY {Carbonell & Gil, 1988} and LIVE {Shen & Simon, 1989}. PRODIGY discovers missing conditions on operators as well as their subgoal ordering of application, and LIVE can extend its concept description (by matching domain functions over the whole experiment in order to find missing aspects of the concept description) when the operators fail to predict an outcome, thus causing the operator to split on the precondition. However, both these systems obtain their guidance from a predefined goal to be obtained, and other domain knowledge.

In this paper we describe learning of a set of concepts representing a procedure. This set of concepts is organised as a hierarchy with higher level concepts referring to lower level concepts. For each new concept identifier (supplied with examples) a hierarchy is built, and when learnt can be used to recognise an example of that procedure, or can be executed to perform that procedure. The *seed* concept is the unique concept at the very top of a hierarchy. Constructive induction is performed on different parts of the hierarchy during an experiment to perform and track execution of the seed concept. Using only imitation and the environment as guiding influences, we show that effective learning in the absence of background knowledge can occur.

After a description of the representation and terminology used, the overall learning strategy is described. Then, details of tracking experiments on concepts, and regeneralisation on failure of expected outcomes, is followed by the generalisation mechanism. After this, two traces of concepts learnt by CAP are examined, followed by discussion of the system learning these and other procedures.

2 Learning Procedures

The representation used for concepts is a subset of Horn clause logic[1] with the following variations. For the purpose of describing temporal concepts it is useful to distinguish two semantic interpretations of predicates: *state predicates* $p(t_1,\ldots,t_n,Time,Time)$, indicate that $p(t_1,\ldots,t_n)$ is true at state *Time* (shown by $p(t_1,\ldots,t_n)$ **at** *Time*), and *action predicates* $p(t_1,\ldots,t_n,Start,Finish)$, indicate that $p(t_1,\ldots,t_n)$ is true during the states *Start* until *Finish* inclusive (shown by $p(t_1,\ldots,t_n)$ **during** *Start / Finish*). Possible interpretations for $A \leftarrow B,C$ are "if B and C are recognised, then we recognise A as well" and "to perform A then one must perform B and C".

The components of the system and the information flow between them is shown in Figure 1. CAP is connected to a simulation which receives primitive actions to be performed and produces predicates describing properties, relations, and actions for the next state. The system starts by observing (ie. recording) all primitive predicates produced by the connected simulation during execution of an example provided by an external agent. This is specified as a list of actions on objects, and is performed by the learner in the simulation. Neither the substructure of this example nor the objects relevant to the intended concept to be acquired are given, merely an observed sequence of states and actions with possibly many irrelevant objects. The intelligent agent that provided the example can never give feedback to the learner as to the nearness to the intended target concept. Thus the example is referred to as an instance of a *seed* concept, ie. one to stimulate and direct search for useful concepts.

In order to create the initial procedure description the complete observed example trace is divided by two levels of division and "irrelevant" predicates trimmed out. A concept hierarchy with one "root" is then generated with all constants automatically variablised. As we are concerned with the active experimentation in this paper, we briefly sketch this process.

Major divisions occur when primitive actions cease acting on one object set and start acting on a different set of objects. Minor divisions occur when one primitive action ceases on an object set and a different primitive action starts on the same object set. Trimming chooses the most changing object and objects from one or two next level sets of object-changing-predicate counts. This is a generalisation as conjoined predicates are removed thus increasing the coverage or use of the resulting description. The justification for this comes from a heuristic of "localised focussing". Just as a child focusses on a subset of all the objects available in order to build an arch, so does CAP.

Example 1

For a seed called do_a_pour the observed sequence is pouring water from a full cup to an empty cup:

```
cup(a) at 1,   cup(b) at 1,   cylinder(c) at 1,   bowl(e) at 1,
   contains-liquid(a) at 1,        not-contains-liquid(b) at 1,
not-contains-liquid(c) at 1,       not-contains-liquid(e) at 1,
                pour(a, b) during 1 / 2,
cup(a) at 2,   cup(b) at 2,   cylinder(c) at 2,   bowl(e) at 2,
not-contains-liquid(a) at 2,        contains-liquid(b) at 2,
not-contains-liquid(c) at 2,       not-contains-liquid(e) at 2
```

The trivial seed concept hierarchy from this, prior to all constant terms replaced by variable terms, is

```
Sorted count list [(b , 3), (a , 3)]
Trimming to objects: [b, a]. Each state trimmed from 8 to 4
    do-a-pour(a, b) during 1 / 2 :-
            sdo-a-pour(b, a) during 1 / 2
    sdo-a-pour(b, a) during 1 / 2 :-
            ssdo-a-pour(b, a) during 1 / 2
    ssdo-a-pour(b, a) during 1 / 2 :-
            cup(a) at 1,
            cup(b) at 1,
            contains-liquid(a) at 1,
            not-contains-liquid(b) at 1,
            pour(a, b) during 1 / 2,
            contains-liquid(b) at 2,
            not-contains-liquid(a) at 2,
            cup(a) at 2,
            cup(b) at 2
```

If at some later stage (maybe after some learning) more examples are observed, they are attempted to be recognised with the current hierarchy. From this, intra-construction forms a new 'root' based on a simplified maximal subsequence match[2].

A cautious learner in an unknown environment will not do the most radical changes to its current concept description and hope for success. Identification of the cause of a failure is easier with smaller numbers of simultaneously tested changes. During execution of a concept hierarchy many concept bodies are being executed. With generalisations possible on all these bodies the amount attempted must be strictly controlled. To this end *generalisation levels* are defined ordered on the least amount of new coverage added to memory, and on minimal invention of new symbols.

After the examples are read in to form the initial hierarchy the system attempts to learn a more general hierarchy using Algorithm 1. But first the seed concepts are assigned initial generalisation levels of *repeat* which force an initial execution of the concept without attempting any generalisations.

Algorithm 1 Learn

1. Find a seed concept $S \leftarrow B$ that has not stopped testing, with the lowest generalisation level L. If there is more than one seed concept at this level then choose the one with the least recent experiment. Bind the starting state term of S to the current simulation state number.
2. Call track(S, L) which executes the whole procedure (ie. concept hierarchy) by executing a body of S which in turn executes each of the components of that body and so on. The generalisation level L represents the *maximum* generalisation to attempted on any body executed.
3. If track exits then S was performed successfully, so the generalisations performed on any part of the hierarchy during execution (if any) are adopted. If no generalisation was tested then the learner tries being more adventurous by increasing the

[1] The reader is referred to {Muggleton&Buntine, 1988}.

[2] These aspects of example observation along with other details are described in detail in {Hume, 1990}

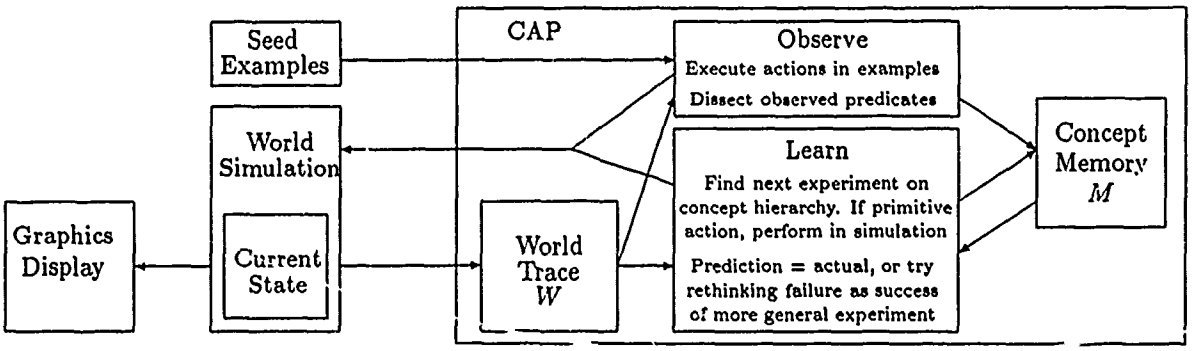


Figure 1: System Diagram and Information Flow

generalisation level of this seed concept. If any generalisation required invention of a new concept then restart any "stopped testing" seed concepts. Continue at Step 1.

4. If track fails and the mistrack occurred in the very last state, then we can be sure of exactly what generalisation on a concept body caused the failure. Record this as a *failed generalisation* (a generalisation never to be attempted again) in that concept. Continue at Step 6.

5. Otherwise, the track failed and we can not identify the exact cause. Mistracking could not be corrected by the generalisation level so *regeneralise* by temporarily increasing the generalisation level[3] and continuing at Step 2 (with original starting state number). If we cannot temporarily increase the level then proceed at Step 6.

6. Because the track has failed then the learner tries being lucky (or more desperate) looking for a more general concept that may be testable in the world, by increasing the generalisation level. But if there are no more generalisation levels then experimentation stops (marking this seed concept as "stopped testing") until new concepts arise due to other seed concepts. Continue at Step 1.

Failure of experiments is critical to specific-to-general learning, otherwise the concepts would never stop being generalised. In Step 4 of the above algorithm the experiment fails in the very last state of the seed concept. If no generalisation was tested (as in a *repeat* experiment) then the existing concept description is incorrect and thus a previous generalisation was wrong. Because the current set of generalisations on a concept have each generalisation applied and tested at a different time, then identification of the exact generalisation causing this failure is impossible. Thus *all* generalisations ever conducted on the concept are undone and the conjunction of them all is recorded as a failed generalisation.

Similarly, when a current generalisation being tested is deemed to be responsible for the failure it is the *conjunction* of all the generalisations that has failed, and not allowed to be tested again.

[3]The ordering is *repeat*, *absorb* and *intra-construct*.

Tracking a predicate attempts to complete execution of that predicate by either showing that it exists in the world (for primitive state predicates), by performing that predicate in the simulation (for primitive action predicates), or by finding the body of that predicate from the concept hierarchy and executing that body or an attempted generalisation of that body:

Given a partially grounded predicate P and a *requested* generalisation level L, we generalise and track the predicate using Algorithm 2:

Algorithm 2 **Track**(P, L)

- If P is primitive then completed execution occurs if $P \in$ world W, or P can be, and is, performed in the simulation. (The performance will add P and the simulation's succeeding state predicates to W.) Fail otherwise.
- For P non-primitive, on it attempt to find an experiment test, T, for a generalisation of the requested level, **generalise**(L, P, T) (described in Algorithm 3, below), or fail.

  With the returned experiment body T, track each of the component predicates $P_i \in T$, in order: **track**(P_i,L). Succeed if all components complete execution, otherwise backtrack to **generalise**(L, P, T), to possibly produce a new experiment the components of which may succeed.

When a concept learner has too specific a concept description, it must generalise that description to cover more positive examples while still excluding negative examples. We have just seen how a test of the new description created is verified. Now, we describe how to create experiments and how to repair ones that fail, so that execution of the overall hierarchy may continue.

Given a *requested* generalisation level L and a partially grounded predicate P, we specify how to return a set of predicates T, representing a test of a generalisation of P with Algorithm 3:

Algorithm 3 **Generalise**(L, P, T)

1. Since P is guaranteed to be a non-primitive predicate find a matching concept $P \leftarrow C$ in memory.
2. Find an experiment of the requested generalisation level L, for P on C returning the test of that experiment, T, and a set of generalisations, G, being tested, ensuring that the union of G and the existing generalisation set on C is not in the failed

generalisations of C. Failing this, attempt to find the most minimal generalisation experiment possible, up to level L. Otherwise try to find another body C in Step 1 and continue there, or fail.

3. If the experiment test, T, fails to track then *regeneralise* by trying to find a different experiment test, T', for a *consistent* generalisation on C. If this cannot be found, try to find another body C in Step 1 and continue there, or fail.

The key to determining success of experiments of generalisation tests is a *consistent* generalisation. A consistent generalisation exists on a concept body, C, when the generalisation list, GL, for C, containing successively applied generalisation sets, has each succeeding generalisation set a *time invariant* subset (a subset up to renaming of time stamps), of the preceding generalisation set in the list.

The constructive induction used for generalisation is *absorption* and *intra-construction* as described in {Sammut, 1981, Sammut & Banerji, 1986, Muggleton & Buntine 1988}, except that *ideal* tests of generalisations have been modified to generate *performable* tests given the current state and execution of the simulated world. The problem is, how can we find a generalisation on a concept body of a predicate being tracked which can be tested given the current execution of the world? The answer depends on how much the world and the concept body must match, and this possible restriction must be taken into account when deciding what generalisation is best to test. Tests of requested generalisations are constructed on an existing concept $P \leftarrow C$ in the following manner:

L = **repeat**
: No generalisation is to be tested, $G = \emptyset$. Return a test equal to the concept body, $T = \{C\}$.

L = **absorb**
: If we think of C as composed of two parts B_1 and B_2 and we have a concept $H \leftarrow B_2$, then *absorption* replaces the existing concept by $P \leftarrow H, B_1$. The generalisation set G is $\{H \leftarrow B_2\}$, and the test of this absorption is $T = \{T_1, B_1\}$ where $H \leftarrow T_1$ is another concept.

L = **intra-construct**
: For *intra-construction*, we have a second existing concept $P \leftarrow B_2, B_3$, and the two concepts are replaced by $P \leftarrow H, B_2$ where H is a new predicate. Also, two auxiliary concepts are invented, $H \leftarrow B_1$ and $H \leftarrow B_3$. The generalisation set G is $\{H \leftarrow B_1\}$, and the test is simply the second concept body, $T = \{B_2, B_3\}$.

In summary, two abstract influences guide the learning process. At a high-level *regeneralisation* represents a tendency in the learner to rethink a mistrack of one more general concept, as success of a different more general concept. The low-level influence is *consistency* recognising justifiably acceptable completed executions of actions performed in the simulation given that certain generalisation tests are current.

3 Examples

Two different sessions are presented here to illustrate aspects of CAP's learning behaviour. The first illustrates the complicated hierarchial nature of the concept learning on a single seed concept, `belayed_climb`. The second displays the autonomous discovery aspect while learning two different seed concepts, `do_a_pour` and `do_a_fillfromtap` alternately in the same session.

In the domain of mountaineering, the world simulation provides primitive properties (`person`, `camp_site`, `summit`, `rope_attached`, `no_rope_attached`), relations (`above`, `level_above`, `at_rope_ends`, `not_at_rope_ends`), and actions (`climbup`, `climbdown` and `sleep_overnight`). The first example shown to CAP starts in state 1 with persons `a` and `b` attached to the rope and level with camp site `camp0`, and `camp1` three rope lengths above `camp0`. Climber `a` lead climbs up until reaching the rope end, and stops. He then belays `b` up. Immediately `b` leads up until the rope end is reached, then he stops. Next, `b` belays `a` up and when `a` leads through and reaches the rope end he is level with `camp1`. Finally `b` is belayed up until level with `a` and the camp site, `camp1`. The grounded (prior to variablisation) seed concept produced from this observation is[4]:

```
belayed-climb(camp0, b, a, camp1) during 1 / 7 :-
    one-move-climb-from-camp(a, b, camp0) during 1 / 2,
    two-move-climb-from-camp(b, a, camp0) during 2 / 4,
    two-move-climb-to-camp(a, b, camp1) during 4 / 6,
    one-move-climb-to-camp(b, a, camp1) during 6 / 7
```

The remainder of the hierarchy is not shown here for lack of space. The second example is the same except that the two camps happen to be an extra rope length apart in height. This means that `a`'s second climb both starts and finishes not level with a camp[5]. Because part of the second example shown to CAP exactly matches (ie. is recognised by) part of the first example then the grounded seed concept `belayed_climb` description is generalised by intra-construction to:

```
belayed-climb(camp0, b, a, camp1) during 9 / 17 :-
    one-move-climb-from-camp(a, b, camp0) during 9 / 10,
    two-move-climb-from-camp(b, a, camp0) during 10 / 12,
    two-move-or-move-to-camp(camp1, a, b) during 12 / 17
```

with the new, invented concepts automatically justified by the examples to produce the following grounded (prior to variablisation) description:

```
two-move-or-move-to-camp(camp1, a, b) during 12 / 17
        two-move-climb to-camp(a, b, camp1) during 12 / #8,
        one-move-climb-to-camp(b, a, camp1) during #8 / 17
two-move-or-move-to-camp(camp1, a, b) during 12 / 17 :-
        two-move(a, b) during 12 / 14,
        two-move-climb-to-camp(b, a, camp1) during 14 / 16,
        one-move-climb-to-camp(a, b, camp1) during 16 / 17
```

With the camps now yet another rope length apart, learning commences with the system merely attempting to perform a repeat experiment from state 18. As the objects involved in the experiment are unknown, the arguments to `belayed_climb` are variables. The trace of the call to `belayed_climb` follows:

```
Tracking 'belayed-climb(CampX, Person1, Person2, CampY) during
    18 / EndTime' at a requested generalisation level of 'repeat'
```

[4] *Variables* are shown by leading uppercase characters, and *constants* by leading lowercase characters or numerals

[5] This is why a camp doesn't appear in `two_move`.

To track and execute this, CAP tracks each of the components of the body of belayed_climb given the starting state 18:

```
tracking one-move-climb-from-camp(Person1, Person2, CampX)
during 18 / T1
tracking two-move-climb-from-camp(b, a, camp0) during 19 / T2
```

After execution of one_move_climb_from_camp some of the variable terms have been unified with constants: Person1/b, Person2/a, CampX/camp0, and T1/19 but CampY/CampY is still variable. Now the invented concept two_move_or_move_to_camp is tracked by trying the first disjunct body two_move_climb_to_camp to see if a can climb straight to the unknown camp CampY:

```
tracking two-move-or-move-to-camp(CampY, a, b) during 21 / T1
tracking two-move-climb-to-camp(a, b, CampY) during 21 / T2
```

This fails since the camp is still two rope lengths above a. CAP then retries tracking the second disjunct body of two_move_or_move_to_camp as shown below:

```
tracking two-move-or-move-to-camp(CampY, a, b) during 21 / T1
tracking two-move(a, b) during 21 / T2
tracking two-move-climb-to-camp(b, a, CampY) during 23 / T1
Do: climbup(b) during 24 / 25, failure.
:: Regeneralising - backtrack looking for a successful retry
```

but finds that after those two climbing moves by a, and another two moves by b, that b is not level with the unknown camp either. Thus a repeat of the existing concept will not allow the procedure to complete execution. By increasing the requested generalisation level to *absorb*, the learner can try to find such a generalisation to enable completed execution of the belay climbing procedure. An absorptive generalisation *is* found on the bodies of two_move_or_move_to_camp that creates the following test of that absorption:

```
two-move(a, b) during 21 / T2,
two-move(b, a) during T2 / T3,
two-move-climb-to-camp(a, b, CampY) during T3 / T4,
one-move-climb-to-camp(b, a, CampY) during T4 / T1
```

This test requires another two_move to be executed by b, thus matching the actual fact that he was not at a camp at the end of his second climb. Thus two_move(b, a) during 23 / 25 exits, execution continues with two_move_climb_to_camp and a finally arriving level with camp1 in state 27:

```
tracking two-move-climb-to-camp(a, b, CampY) during 25 / T4
tracking one-move-climb-to-camp(b, a, camp1) during 27 / T1
```

Finally the procedure belayed_climb exits after having to allow an absorptive generalisation on two_move_or_move_to_camp in order for the procedure to complete execution:

```
Tracked success for 'belayed-climb(camp0,b,a,camp1) during 18/28'
at requested level 'repeat' and actual level 'absorb' taking 28.38 secs
```

Because of the success, any generalisations are confirmed, including the absorption and a number of unplanned low-level intra-constructions on bodies of two_move and others (not shown here). Two of the completed grounded (prior to variablisation) new concept descriptions are shown for brevity (stwo_move is a recursive sub-concept of two_move):

```
two-move-or-move-to-camp(camp1, a, b) during 21 / 28 :-
        two-move(a, b) during 21 / 23,
        two-move-or-move-to-camp(camp1, b, a) during 23 / 28
stwo-move(b, a) during 23 / 25 :-
        symmet-at-rope-ends(a, b) at 23,
        rope-attached(b) at 23,
        rope-attached(a) at 23,
        person(b) at 23,
        person(a) at 23,
        above(a, b) at 23,
        climbup(b) during 23 / 24,
        stwo-move(b, a) during 24 / 25
```

and the two grounded invented concepts:

```
symmet-at-rope-ends(a, b) at 23 :- at-rope-ends(b, a) at 23
symmet-at-rope-ends(a, b) at 23 :- at-rope-ends(a, b) at 23
symmet-not-at-rope-ends(a,b) at 24 :- not-at-rope-ends(b,a) at 24
symmet-not-at-rope-ends(a,b) at 24 :- not-at-rope-ends(a,b) at 24
```

Thus by observing two examples of a belayed climb from one camp to another, and then faced with a situation that exact imitation would fail to succeed, the learner has acquired the ability to climb any two persons from one camp to another camp in safety.

The second example involves concepts of water transfer and containment. Here, the world simulation provides primitive properties (cup, bowl, cylinder, contains_liquid and not_contains_liquid) and primitive actions (fillfromtap and pour). The initial state consists of four objects, two cups, a closed cylinder, and a bowl. The first cup contains water.

The first seed example is the single action pour(a, b) during 1 / 2 pouring the contents of cup a to cup b. The initial seed concept hierarchy from this obser-

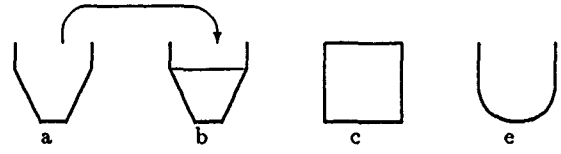


Figure 2: World simulation in state 2

vation is given in Example 1. The second seed concept example do_a_fillfromtap consists of the one action fillfromtap(a) during 2 / 3, filling cup a from an invisible tap. The second grounded seed concept

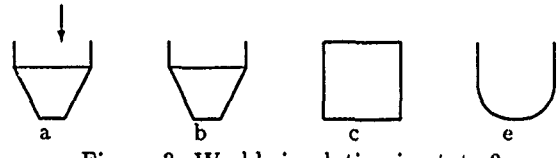


Figure 3: World simulation in state 3

hierarchy is given by :

```
Sorted count list [(a , 3)]
Trimming to objects: [a]. Each state trimmed from 8 to 2
    do-a-fillfromtap(a) during 2 / 3 :-
            sdo-a-fillfromtap(a) during 2 / 3
    sdo-a-fillfromtap(a) during 2 / 3 :-
            ssdo-a-fillfromtap(a) during 2 / 3
    ssdo-a-fillfromtap(a) during 2 / 3 :-
            not-contains-liquid(a) at 2,
            cup(a) at 2,
            fillfromtap(a) during 2 / 3,
            contains-liquid(a) at 3,
            cup(a) at 3
```

Now that these two seed concept hierarchies have been created, the program commences the learning phase. Randomly, do_a_pour is attempted to be proved in the world with an initial requested generalisation level of *repeat*, (ie. no generalisation attempted) and the starting state term bound to the current state number 3. Thus, the body of ssdo_a_pour has a substitution

are variablised and stored even though the generalisation for which they were constructed failed. This is to prevent the same intra-construction from being repeated in the future.

The failure means that this seed concept's generalisation level is reset to *repeat* to rebuild confidence in the current description. As this level is less than that of `do_a_pour` then a `repeat` experiment `fillfromtap(b)` `during 11/12` is attempted on `do_a_fillfromtap`. This completes successfully, increasing the generalisation level to `absorb`. As this is still lower than the level of `do_a_pour` then the experimentation continues with the same seed concept.

Although an absorption can be tested, it is equal to the only member of the failed generalisations of `ssdo_a_fillfromtap`. A repeat experiment `fillfromtap(b)` `during 12/13` is thus conducted. The successful completion would increase the generalisation level to `intra-construct` but as this level is not allowed `do_a_fillfromtap` is marked as stopped. Experimentation continues with `do_a_pour` requesting an intra-construction.

An existing concept `cuporcylinder` means that an absorption would be planned if the cylinder was chosen as the destination. Thus the bowl is chosen for the planned intra-construction. Pouring from cup to bowl is performed `pour(b,e)` `during 13/14`. The resulting state does not match the test but, regeneralisation finds a consistent generalisation on the the body equal to the restamped initiating generalisation. The new grounded intra-constructed concepts are:

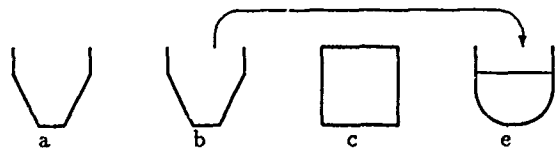


Figure 8: World simulation in state 14

concave-up(e) at 13 :- [cup(e) at 13]
concave-up(e) at 13 :- [bowl(e) at 13]

The success means that this seed concept's generalisation level is reset to repeat, and any "stopped testing" concepts unmarked, because a new concept was invented which may mean that other generalisations may now be testable. As this level is less than that of `do_a_fillfromtap` then a `repeat` experiment is now attempted on `do_a_pour`: The system selects the first two cups for the repeat experiment `pour(b,a)` `during 14/15`. The fact that they happen to be both empty satisfies the concept. After pouring, there is a test predicate not satisfied by the world, `contains_liquid(a)` `at 15` and a world predicate not covered by the test, `not_contains_liquid(a)` `at 15`. Thus the experiment must fail. As no generalisation was tested we know that some subset of the current generalisations is incorrect. As we do not know which one, the *conjunction* of all the current generalisations on the body is recorded as a failed generalisation. We also undo those generalisations thus unwinding the body back to the originally observed example. The unsuccessful completion of this experiment increases the generalisation level to `absorb` in the hope that a more general test may possibly succeed.

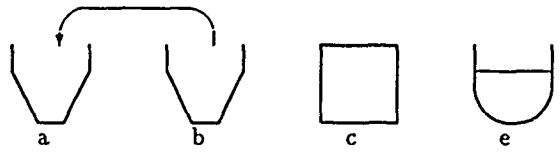


Figure 9: World simulation in state 15

Although the other seed concept `do_a_fillfromtap` is at a level of `absorb` experimentation remains with `do_a_pour` due to a delay in restarting stopped concepts. In this situation, two simultaneous absorptions are found on different parts of the restricted body - one testing a change in the source object from the cup to a bowl and the other testing a change in the destination object from a cup to a cylinder. After

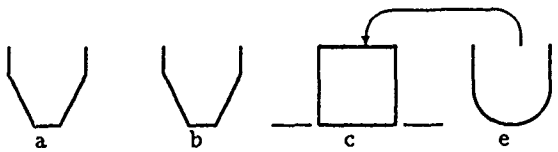


Figure 10: World simulation in state 16

`pour(e,c)` `during 15/16` there is a test predicate that is not satisfied by the world `contains_liquid(c)` `at 16` and a world predicate not covered by the test, `not_contains_liquid(c)` `at 16`. Thus the experiment must fail, with the generalisation marked as a failed one. The generalisation level does not increase since some knowledge (a failed generalisation) was learned.

Experimentation alternates to `do_a_fillfromtap` at the same level of `absorb`. The only predicate in the starting state of `ssdo_a_fillfromtap` that can have an absorption constructed on it is `cup`. As there is a current generalisation `maybe_contains_liquid` on that restricted body and the combination of this and `cylinder` from `cuporcylinder` is marked as a failed generalisation, then we try the combination of `maybe_contains_liquid` and `bowl` from `concave_up` as a planned absorption. Here, a following generalisation is

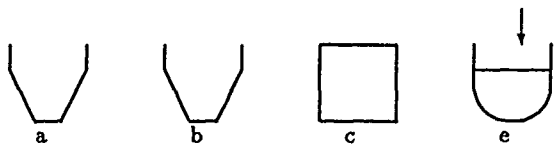


Figure 11: World simulation in state 17

required which is consistent with the initiating generalisation, and therefore this is a successful experiment, and the generalisation level is unchanged. The new grounded concept description is:

```
ssdo-a-fillfromtap(e) during 16 / 17 :-
        concave-up(e) at 16,
        maybe-contains-liquid(e) at 16,
        fillfromtap(e) during 16 / 17,
        concave-up(e) at 17,
        contains-liquid(e) at 17
ElabJustifications:
  initiating(maybe-contains-liquid(e) at 16,
     [not-contains-liquid(e) at 16], [contains-liquid(e) at 16]),
  initiating(concave-up(e) at 16, [cup(e) at 16], [bowl(e) at 16]),
  following(concave-up(e) at 17, [cup(e) at 17], [bowl(e) at 17])
FailedGens:
  [maybe-contains-liquid(e) at 16 - [contains-liquid(e) at 16],
   cuporcylinder(e) at 16 :- [cylinder(e) at 16]]
```

searched for that requires no generalisation on that body. Unfortunately this is not possible as both the cups are full of water (the observed example had one cup full and one empty). Thus a minimal unplanned generalisation is found by intra-construction covering the unmatched predicate `not_contains_liquid(b) at 3`. This represents a test of whether or not the destination cup need be empty at the start of the experiment. Thus the primitive action `pour(a,b) during 3/TEnd` is performed in the simulation. The succeeding state Figure 4 is as predicted by the experiment and

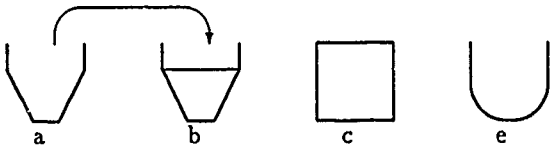


Figure 4: World simulation in state 4

thus the generalisation is adopted. The new grounded intra-constructed concepts are:

maybe-contains-liquid(b) at 3 :- [not-contains-liquid(b) at 3]
maybe-contains-liquid(b) at 3 :- [contains-liquid(b) at 3]

The generalisation level for `do_a_pour` does not increase as learning occurred from the experiment.

The other seed concept `do_a_fillfromtap` is at the same generalisation level and a repeat experiment is performed on it as there is an unfilled cup. The generalisation level for `do_a_fillfromtap` increases to `absorb` as nothing was learned. This increase corresponds to confidence in the existing seed concept hierarchy stimulating experiments for a more general description.

`Do_a_pour` is attempted next as it is at the lower generalisation level `repeat`. This experiment `pour(b,a) during 5/6` is successful and the generalisation level increases as nothing was learned. Since both seed concepts are at the next planned generalisation level of `absorb` either is selected for the next complete experiment. With `do_a_pour`. CAP explicitly tests whether or not the object from which the water comes must contain liquid at the start of the action. Thus it pours an empty cup `b` into a full cup `a`, `pour(b,a) during 6/7`. Again the succeeding state needs no generalisa-

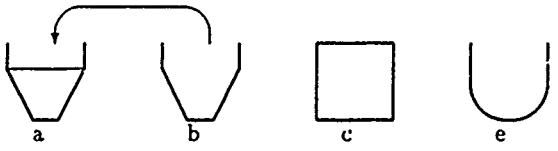


Figure 5: World simulation in state 7

tion inorder to be proved and thus the generalisation is confirmed. As learning occurred, the generalisation level is unchanged.

Although this generalisation was correctly tested, the resulting description is not in the intended meaning of the trainer's example of `do_a_pour`. Pouring an empty cup into an empty cup would have not succeeded as a different generalisation would be required to track completion, and thus inconsistency.

But now, because both seed concepts are at the same generalisation level, an explicit absorption is attempted on the concept hierarchy for `do_a_fillfromtap`.

With the body of `ssdo_a_fillfromtap` a substitution is searched for that requires an absorptive generalisation. This is possible as cup `a` is full of water. This generalisation means that it doesn't matter whether or not the cup you are filling already contains water, afterwards it is guaranteed to contain water. The experi-

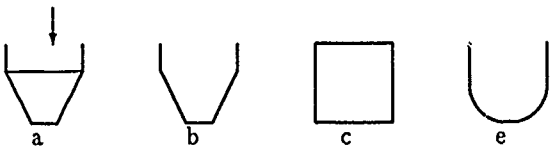


Figure 6: World simulation in state 8

ment `fillfromtap(a) during 7/8` is a success, the generalisation adopted, and the generalisation level unchanged.

Because both seed concepts are still at the same generalisation level, they are attempted alternately. Thus, `do_a_pour` is now tracked for a requested absorption. As no more generalisations are possible on the starting state[6] of `ssdo_a_pour`'s body then a repeat experiment is performed. Since nothing was learned during this successful experiment `pour(b,a) during 8/9` the generalisation level for `do_a_pour` increases. But planned intra-construction is not allowed, thus testing on this seed concept hierarchy is marked as stopped. Similarly, the other seed concept `do_a_fillfromtap` is marked as stopped, after a repeat experiment `fillfromtap(b) during 9/10`.

Although planned intra-construction is not allowed when learning is exhausted in the world the learner may conduct a once-off explicitly planned for invention on each of the seed concepts. Learning temporarily recommences at this level for both seed concepts, with `do_a_fillfromtap` arbitrarily chosen first.

Here, `ssdo_a_fillfromtap` has a choice of substitutions that require an intra-constructive generalisation. This is possible with the test having either a cylinder or a bowl replacing the cup being filled. The cylinder `c` is selected as we can see in Figure 7 there is no water in the cylinder

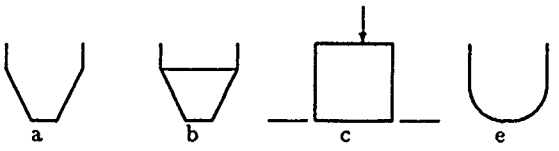


Figure 7: World simulation in state 11

after the action `fillfromtap(c) during 10/11`. Although intra-construction could allow the remainder of `ssdo_a_fillfromtap` to track, the extra construction would cause an inconsistent generalisation. This failed generalisation test actually occurs in conjunction with the previous generalisation `maybe_contains_liquid(c) at 10 ← contains_liquid(c) at 10`, and it is the *conjunction* of this with `cuporcylinder(c) at 10 ← cylinder(c) at 10` that is recorded as having failed.

The new grounded intra-constructed concepts

cuporcylinder(c) at 10 :- [cup(c) at 10]
cuporcylinder(c) at 10 :- [cylinder(c) at 10]

[6] Actually defined as an *restriction* in {Hume, 1990}.

Informally, this final resulting concept description for do_a_fillfromtap means that given a concave upwards object, whether full of water of not, the result of filling it from a tap means it becomes full of water.

The ElabJustifications is a recording of the justification (replacing predicate, the replaced predicates, and the test of replacing predicate) for each generalisation on the body. FailedGens represents the set of conjunctions of generalisations on the original concept body that have been tested as being outside the correct description for this concept.

Learning switches to do_a_pour with absorb. In this situation, two simultaneous absorptions are found on different parts of the restricted body - one testing a change from a cup to a cylinder and the other testing whether the source object need contain water at the start. The action performed is pour(b,c) during 17/18. After pouring, there is a test predi-

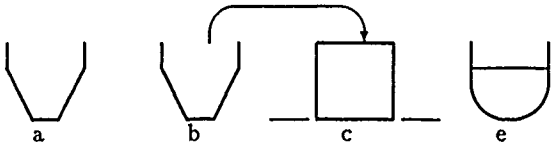


Figure 12: World simulation in state 18

cate not satisfied by the world contains_liquid(c) at 18, and a world predicate not covered by the test not_contains_liquid(c) at 18. This means that the experiment must fail. From an observer's point of view it would have failed due to either of these elaborations alone, an empty source or an unfillable destination. We mark this as another failed generalisation. The generalisation level does not increase since some knowledge (a failed generalisation) was learned.

Experimentation alternates to do_a_fillfromtap at the same level of absorb. But, after a repeat experiment fillfromtap(a) during 18/19 it is marked as stopped as the generalisation level cannot increase.

Experimentation continues with do_a_pour. Two simultaneous absorptions (different to any of the failed generalisations) are found on different parts of the restricted body - one testing a change from the destination cup to a bowl and the other testing whether the destination object need contain water at the start. The action performed is pour(a,e) during 19/20. The

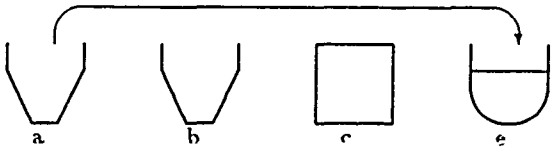


Figure 13: World simulation in state 20

resulting situation requires a following generalisation which is equal to the initiating generalisation except for the substitution. This satisfies consistency and thus is a successful experiment. The generalisation level remains unchanged and as the other seed concept is stopped we continue with do_a_pour.

Here, an absorption is found on the restricted body testing a change from the source object being a cup to being a bowl. The action performed is this case is

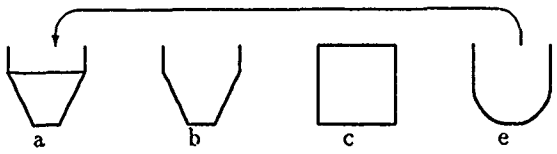


Figure 14: World simulation in state 21

pour(e,a) during 20/21. Again, a following generalisation is required which is consistent and therefore this is a successful experiment. The new grounded concept description is:

```
ssdo-a-pour(a, e) during 20 / 21 :-
        concave-up(e) at 20,
        concave-up(a) at 20,
        maybe-contains-liquid(a) at 20,
        contains-liquid(e) at 20,
        pour(e, a) during 20 / 21,
        concave-up(e) at 21,
        concave-up(a) at 21,
        contains-liquid(a) at 21,
        not-contains-liquid(e) at 21
ElabJustifications:
  initiating(concave-up(a) at 20, [cup(a) at 20], [bowl(a) at 20]),
  initiating(maybe-contains-liquid(a) at 20,
      [not-contains-liquid(a) at 20], [contains-liquid(a) at 20]),
  following(concave-up(a) at 21, [cup(a) at 21], [bowl(a) at 21]),
  initiating(concave-up(e) at 20, [cup(e) at 20], [bowl(e) at 20]),
  following(concave-up(e) at 21, [cup(e) at 21], [bowl(e) at 21])
FailedGens:
  [cuporcylinder(a) at 20 :- [cylinder(a) at 20],
   maybe-contains-liquid(e) at 20 :- [not-contains-liquid(e) at 20]],
  [cuporcylinder(a) at 20 :- [cylinder(a) at 20],
   concave-up(e) at 20 :- [bowl(e) at 20]],
  [maybe-contains-liquid(a) at 20 :- [contains-liquid(a) at 20],
   maybe-contains-liquid(e) at 20 :- [not-contains-liquid(e) at 20],
   concave-up(a) at 20 :- [bowl(a) at 20]]
```

Informally, this final resulting concept description for do_a_pour means that given an source concave upwards object containing water and a destination concave upwards object maybe containing water, the result of pouring means that the destination is full of water and the source is empty.

As no further generalisations are possible on the concept, a repeat experiment, pour(a,b) during 21/22 is performed. As the generalisation level cannot be increased, testing is stopped.

4 Discussion

Other concepts which CAP has learned displaying different aspects of its behaviour, include:
Climbing to summit and back with overnight camps (uses belay_climb),
Building a arch of any height (invented level_or_above, symmetric not_touching and level_above, and two new movement disjuncts).

In these examples (except water pouring) CAP could only learn the most useful concept if there was a likely repeated pattern to the actions in the intended concept description.

In water pouring, a single action constitutes the concept and the system discovers and invents concepts to generalise preconditions for the action from just one observed example. Thus for single action procedures, CAP and LIVE {Shen & Simon, 1989} perform and discover equivalently (but with different representations) given no domain knowledge. This is because in LIVE one can define an empty goal causing the sys-

tem to merely predict the effect of an action given any state.

The mountaineering example of climbing to the summit with overnight stops and returning, illustrates the system's ability to use concepts that it has previously learned in order to describe new concepts. Here, `belay_climb` was successfully recognised and used in moving between camps.

The *building an arch* example displayed one problem symptomatic of the system's tendency to generalise state predicates in order to execute actions. In this case learning an arch from blocks and a cross beam required aligning the two base blocks just inside and infront of the ends of the beam, then building the columns up equally and putting the beam on top of the columns. If, during experimentation, only a short beam is available that causes the columns of blocks to touch, then a concept `maybe_touch(a,b) at 10` will be invented to cover the difference between the existing predicate `not_touching(a,b) at 10` and the current predicate `touching(a,b) at 10`. When the generalisation causing this is tested, the rest of the experiment succeeds, thus incorrectly accepting it as a valid generalisation. This particular problem would be remedied by actually *using* the arch for the purpose for which it was constructed, ie. by passing another object *through* the constructed arch. But as this would have to appear as a part of the example, every use of a concept for constructing objects would need to appear in the examples observed by the system. This is clearly deficient but indicative of the lack of suitability of CAP for goal oriented concepts.

Attempts were also made to learn problem solving tasks such as the tower of hanoi puzzle, genetic inheritance, and lense grinding, but without success. This failure was due to the lack of goal direction in CAP. It simply could not recognise that aspects of the final state were vitally important in learning the required concepts. Some aspects of these concepts were learned where the general *form* of the action ordering in solution paths had common and/or repeated patterns (most success with tower of hanoi). But, in the worst case CAP would have to be shown every solution path.

Another problem is the sensitivity to extraneous details during the observed sequence dissection. If a block is moved left until touching a target block, but it passes a distant block on the way, then three relational predicates change with the distant block (`rightof`, `level_rightof`, `rightof`) but only two with the intended target (`not_touching`, `touching`). In this case a distant object must always be passed on the way to a target block. Thus the "localised focussing" heuristic biases the system to action sequences where the intended relevant objects are those in the most frequently changing predicates.

The implementation consists of about 3200 lines of UNSW Prolog 4.2 excluding comments, the simulations, and the interface thereto. Times for a single experiment (at about 10kLIPS) ranged from 0.5 sec for water pouring to 15 minutes for arch building (due to 120 predicates per state × 55 states to build it).

5 Conclusion

Given only a few examples and no background knowledge or domain theory it is possible to effectively learn in a wide range of interactive environments. A representation and mechanism using first-order logic is largely responsible for this, but at a cost of controlling unconstrained behaviour. In most systems with this breadth of application, the solution to this results in either, a large amount of domain and control knowledge (eg. EBL, problem solving learners), or a large amount of explicit oracle interaction (eg. Marvin, CIGOL).

Without domain knowledge or control and without oracle interaction, effective learning has been demonstrated that uses one very abstract control heuristic, "imitate activity you see in the environment". The categorisation of positive and negative instances of concepts is done by the environment, and the "drive" to learn comes from a continual search to try and match patterns by performing them. Thus, recognisably successful executions of concepts that partially match existing concepts result in more general concepts being adopted.

Additionally, concepts learnt, can be used to assist learning of later concepts since they become part of the description language. Also, discovery of clusters of properties, relations and actions occurs which are grouped by facilitation of valid procedure executions.

5.0.1 Acknowledgments

The author would like to thank Claude Sammut and the referees for their comments on drafts of this paper.

5.0.2 References

P.M. Andreae, *Justified Generalisation: Acquiring Procedures From Examples* MIT AI Lab T.R. 834 (1985)

J.G. Carbonell and Y. Gil, Learning by Experimentation. In R.S. Michalski, T.M. Mitchell and J.G. Carbonell (Eds.), *Proceedings of Fifth International Machine Learning Workshop* Morgan Kaufman (1988)

B. Falkenhainer and S. Rajamoney, The Interdependencies of Theory Formation, Revision, and Experimentation. In R.S. Michalski *et al* (Eds.), *Proceedings of Fifth International Machine Learning Workshop* (1988)

D.V. Hume, *Induction of Procedures in Simulated Worlds* PhD Thesis (forthcoming), Department of Computer Science, University of New South Wales, (1990)

S. Muggleton and W. Buntine, Machine Invention of First-order Predicates by Inverting Resolution. In R.S. Michalski *et al* (Eds.), *Proceedings of 5th IMLW* (1988)

C.A. Sammut, *Learning Concepts by Performing Experiments* Ph.D. thesis, Department of Computer Science, University of New South Wales, November (1981)

C.A. Sammut and R.B. Banerji, Learning Concepts by Asking Questions. In R.S. Michalski, T.M. Mitchell and J.G. Carbonell (Eds.), *Machine Learning: An Artificial Intelligence Approach (Vol. 2)* Morgan Kaufman (1986)

J.W. Shavlik, Acquiring Recursive Concepts with Explanation-Based Learning. *Proceedings of 11th IJCAI* 20-25th Aug., Detroit, Michigan. Morgan Kaufman (1989)

W. Shen and H.A. Simon, Rule Creation and Rule Learning through Environment Exploration *Proceedings of 11th IJCAI* Detroit, Michigan. Morgan Kaufman (1989)

K. VanLehn, Learning One Subprocedure per Lesson. *Artificial Intelligence* 31 pp 1-40 (1987)

Beyond Inversion of Resolution

Céline Rouveirol, Jean François Puget
Université Paris Sud, LRI, Bldg 490, 91405 Orsay, France
email : celine@lri.lri.fr, puget@lri.lri.fr

Abstract

A framework for induction has been proposed in (Muggleton & Buntine, 1988) and implemented in the CIGOL system. We have extended the operators introduced in CIGOL to non unit clauses. Doing this, we have discovered two limitations of inversion of resolution, mainly one about the Absorption operator. We therefore propose a new operator called Saturation that replaces Absorption in our system. We give a clear definition of this operator after a formal analysis of the problem. We then describe how the two other operators, namely Intraconstruction and Truncation, can be reformulated and re-implemented.[1]

1 Introduction

1.1 Motivations

Induction is the process of building a general theory from particular facts. This definition has led to many interpretations in the Machine Learning community, depending on the meaning ascribed to generality. We will not discuss this issue here, and refer to (Plotkin, 1971; Mitchell, 1982; Kodratoff & Ganascia, 1986; Buntine, 1988; Helft, 1988; Kodratoff, 1988; Niblett, 1988; Kodratoff, 1989) for a review of these different definitions.

A very general framework states that A is more general than B if A logically implies B (denoted A |= B). This basically means that the models of A are also models of B; by that, no restriction is made about any inference procedure we may use.

(Muggleton & Buntine, 1988) specializes the previous expression stating that A is more general than B if B can be derived from A using SLD resolution, (denoted A $\vdash_{SLD}$ B). The process of induction can be described in their framework as follows. Given a domain theory T and an example E, such that T $\vdash\!\!\!/_{SLD}$ E (E is not derivable from T using SLD resolution), their aim is to generate a new domain theory T' such that T' $\vdash_{SLD}$ E and T' $\vdash_{SLD}$ T. T' can then be induced from T and E by the process of 'inversion of resolution'. Their system, called CIGOL uses 3 operators : Absorption that inverts one single step of resolution, Intraconstruction that inverts two resolution steps of two clauses with one common clause, Truncation that inverts substitutions.

Finding a general solution to inversion of resolution raises some problems, both from a theoretical and from a practical point of view. Some of these problems are listed in section 2. The first motivation of our work was to find a simple and efficient solution for inversion of resolution when using a first order logic language without function symbols. Besides, we have used an automatic representation change that transforms functions into predicates (by creating one predicate that has one additional argument representing the result of the function) and vice versa, thus enabling us to deal with arbitrary Horn clauses. The conjunction of these two results allows us to extend the three operators introduced in CIGOL to non unit clauses. This work has been implemented in Quintus PROLOG in a system called IRES (Rouveirol & Puget, 1989) and is briefly described in section 2.

With this implementation, we became aware of problems which are not related to our particular solution but rather are fundamental limitations of inversion of resolution. We review these problems in section 3. We also introduce a solution that relates inversion of resolution to results obtained in Logic Programming. We therefore suggest a re-thinking of inversion of resolution operators in section 4 and demonstrate the expected improvements on a small example in section 5. Finally, we conclude on promising research directions in section 6.

1.2 Notations and definitions

We use a subset of first order logic, namely Horn Clauses (as in Prolog) as representation language. We recall here the basic definitions used in Logic Programming, and refer to (Lloyd, 1987) and (Genesereth

[1]The first author has a French MRT scholarship and is partially supported by CEC, through the ESPRIT-2 contract MLT 2154. The second author is supported by PRC-IA.

& Nilson, 1987) for an extended treatment of this subject.

A **predicate** is given by a predicate symbol, and an arity. An **atom** is a predicate applied to terms, i.e., an expression of the following form : $p(t_1,...,t_m)$ where p is a predicate symbol, and the t_i are terms. A **positive literal** is an atom, a **negative literal** is the negation of an atom.

Briefly, a **substitution** is a finite set of pairs x_i/t_i, where x_i is a variable and t_i a term. A substitution σ is applied to a formula F by replacing each occurrence of the variables by the corresponding term, the result is noted $F\sigma$. If $\{(s_i, t_i)\}$ is a set of pairs of terms or a pair of atoms, a **unifier** is a substitution σ such that for every i, $s_i\sigma = t_i\sigma$. A **most general unifier (mgu)** is a substitution σ such that for every unifier θ, there exists a substitution γ such that $\sigma = \theta\gamma$.

A **clause** is a finite disjunction of literals with all its variables universally quantified. A **Horn clause** has at most one positive literal. A **goal clause**, is a clause with no positive literal and is denoted $\leftarrow B_1 \wedge ... \wedge B_m$. A **definite clause** is a clause with *exactly one* positive literal and is noted $P \leftarrow B_1 \wedge ... \wedge B_m$. P is the **head** of the clause, and the conjunction $B_1 \wedge ... \wedge B_m$ is the **body** of the clause. A **unit clause** has an empty body. By **logic program**, we intend a set of definite clauses, i.e., it does not contain any goal clause.

P entail F is noted $\mathbf{P} \models \mathbf{F}$, and means that the formula $F \leftarrow P$ is valid.

In this framework, a concept is represented by a predicate. A clause is interpreted as a concept definition: its head identifies the defined concept, its body lists the conditions for belonging to the concept.

In the remaining of the paper, two definition of generality will be used. The first one is used in CIGOL and the first implementation of IRES : a theory T is more general than a formula F if and only if F can be proved from T, that is $T \vdash_{SLD} F$. In such a case formula F is said to be covered, or explained, by theory T : explanation means proof. The second one is iused in a latter version of IRES : a theory T is more general than a formula F if and only if T entails F, that is $T \models F$. In such a case T also explains F.

1.3 Resolution

We consider in the following special case of resolution, the one used in PROLOG, called **SLD resolution.** It is defined as follows. Let us consider two clauses

(C1): $H_1 \leftarrow A \wedge \alpha$

(C2): $H_2 \leftarrow \beta$

where α and β are conjunctions (maybe empty) of atoms. A is an arbitrary literal of the body of (C1), and a is the remaining of the body of (C1). We say that (C1) can be resolved with (C2) if H_2 and A unifies with mgu θ. We have then $A\theta = H_2\theta$. The result of the resolution step is the new clause:

Res(C1,C2): $H_1\theta \leftarrow \beta\theta \wedge \alpha\theta$

By such definition of resolution, we impose that the resolution step unifies the head of (C2) with a literal within the body of (C1).

Note that (C1) can also be a goal clause, i.e. the clause without head

(C1): $\leftarrow A \wedge \alpha$

in such a case the result of the resolution step is

Res(C1,C2): $\leftarrow \beta\theta \wedge \alpha\theta$

2 Inversion of resolution

2.1 CIGOL

In order to give an hint of the difficulties to face to solve inversion of resolution in the general case, we give, after (Muggleton & Buntine, 1988), the formula used for inverting a single step of resolution in the Absorption operator. If C1 and R are clauses, all clauses C2 such that R is the resolvent of C1 and C2 can be computed with the following:

$$C2 = (R - (C1 - \{L1\}) \cdot \theta1) \cdot \theta2^{-1} \cup \{L2\} \quad (1),$$

where, L1 and L2 are the resolved literals in C1 and C2, $\theta1$ and $\theta2$ are the associated substitutions ($L1 \cdot \theta1 = L2 \cdot \theta2$), and $\theta2^{-1}$ is the 'inverse substitution' of $\theta2$. An inverse substitution basically replaces constants or terms by variables. It can be viewed as an extension of the turning constant into variables generalization rule (Michalski, 1983).

CIGOL only provides a partial solution for this equation, that is it solves this equation in the case where C1 is a unit clause (in other terms, C1 - {L1} is empty). Equation (1) then simplifies to

$$C2 = (R \cdot \theta1) \cdot \theta2^{-1} \cup \{L2\}$$

where the unknowns are $\theta2^{-1}$, $\theta1$ and L2. The main difficulty is to build $\theta2^{-1}$. During the resolution step $\theta2$ substitutes one single variable in C2 with an arbitrary term which is also found in $(R \cdot \theta1)$; conversely, for building $\theta2^{-1}$, CIGOL has to choose which subterms in R to replace by one common variable in C2. There can be a combinatorial explosion at this step. In CIGOL, the choice of this 'inverse substitution' is guided by optimization of information compression when replacing C2 by R (see (Muggleton, 1988) for details about CIGOL strategy).

2.2 IRES

2.2.1 Working hypothesis

We have chosen a different simplifying hypothesis to solve inversion of resolution. We have kept in a first step the same structure as CIGOL (that is, with three operators Absorption, Intraconstruction and Truncation) but we only deal with clauses without function symbols.

This simplifies a lot the descriptions and implementations of the operators. In order to overcome the strong limitation introduced by our working hypothesis, we use an automatic representation change that transforms functions into predicates (**flattening**) and vice versa, without changing the semantics of the overall program. Let us briefly describe this representation change with a small example. The clause to flatten is :

(Ex): $member(blue,[blue])\leftarrow$

The second argument of the predicate member, [blue], is in fact the term *cons(blue,nil)*. The idea of flattening is to replace the function *cons(X,Y)* by a new predicate $cons_p(X,Y,Z)$, where Z is the result of applying cons to X and Y. In our example, this replacement yields the new clause:

(Ex'): $member(blue,Z)\leftarrow cons_p(blue,nil,Z)$

The new predicate $cons_p$ is defined as follows to preserve the semantics of the original clause:

(CONS): $cons_p(X, Y, cons(X,Y))\leftarrow$.

The meaning of the input clause (Ex) has not been changed since the resolution of (Ex') with (CONS) gives (Ex), as can be checked. This is a general property of the flattening transformation. The flattening algorithm performs these transformations for all the occurrences of functions in the clauses, in particular for the constants, since there are functions of arity 0. For instance, *blue* is replaced by a predicate $blue_p(A)$, etc. The final result of flattening is thus the following clause:

(Ex"): $member(X,Z) \leftarrow$
$cons_p(X,Y,Z) \wedge blue_p(X) \wedge nil_p(Y).$

together with the clauses that define the predicates $cons_p$, $blue_p$ and nil_p :

(CONS): $cons_p(X, Y, cons(X,Y))\leftarrow$.
(BLUE): $blue_p(blue) \leftarrow$.
(NIL): $nil_p(nil)\leftarrow$.

This kind of representation transformation is classical in Logic Programming and has been done manually in the system MARVIN (Sammut, 1981). Our originality is to have automatized it.

It is worth noticing that two identical subterms are replaced by a single variable. Here, the *blue* constant which appeared twice, is replaced by the single variable *X*. This choice simplifies a lot the search for inverse substitution in Absorption (Rouveirol & Puget, 1989). By doing this, we stick very much to the example, and we are of course more sensitive to coincidental occurrences of one value in the example. We can get rid of these coincidences by further generalizing the examples when applying Intraconstruction.

The flattening transformation enables to handle arbitrary Horn clauses as input and output for the operators. In practice, it allows us to relax the unit clause assumption put on the input clauses of Absorption, Intraconstruction and Truncation in CIGOL.

2.2.2 A sample session of IRES

Let us illustrate how IRES proceeds to complete the following theory defining family relationships (CIGOL cannot handle this example because of the unit clause restriction).

T_1: *grandfather(X,Z)* $\leftarrow$
father(X,Y) $\wedge$ *father(Y,Z).*

T_2: *father(X,Y)* $\leftarrow$
child_of(Y,X) $\wedge$ *sex(X,male).*

T_3: *mother(X,Y)* $\leftarrow$
child_of(Y,X) $\wedge$ *sex(X,female).*

T_2 and T_3 are flattened into :

T_{2f}: *father(X,Y)* $\leftarrow$
child_of(Y,X) $\wedge$ *sex(X,S)* $\wedge$ *male(S).*

T_{3f}: *mother(X,Y)* $\leftarrow$
child_of(Y,X) $\wedge$ *sex(X,S)* $\wedge$ *female(S).*

Suppose now that the following example is met.

E: *grandfather(tom,liz)* $\leftarrow$
father(tom,helen) $\wedge$ *child_of(liz,helen)* $\wedge$
sex(helen,female).

which yields, once flattened :

Ef: *grandfather(X',Z')* $\leftarrow$
father(X',Y') $\wedge$ *child_of(Z',Y')* $\wedge$
sex(Y',S') $\wedge$ *tom(X')* $\wedge$ *liz(Z')* $\wedge$
helen(Y') $\wedge$ *female(S').*

E is not entailed by the available theory, because *father(helen,liz)* cannot be derived, by resolution with one of the clauses of the domain theory, from *child_of(liz,helen)*, *sex(helen,S)* and *female(S)* , but *mother(helen,liz)* can be derived : the definition of grandfather is clearly incomplete.

In the remaining of the example, IRES proposes to the user all the operators that can be applied at one step, and the user chooses which operator to apply. At this step, all three operators are applicable, the user chooses Absorption, as it generalizes the expression of the example.

2.2.2.1 Absorption

Absorption rewrites one clause C_2 (be it an example or a rule of the domain theory) using a concept defined in another clause C_1. Absorption of a clause C_1 (the absorbed clause) by a clause C_2 (the absorbant clause) is possible if the body of clause C_2 can be unified with a part of the body of C_1.

In our example, there is only one clause that can be absorbed by the example E, namely T_3. This clause can be absorbed by Ef because its body

child_of(Y,X) $\wedge$ *sex(X,S)* $\wedge$ *female(S)* can be unified with the subpart

child_of(Z',Y') $\wedge$ *sex(Y',S')* $\wedge$ *female(S')*

of the body of Ef with the substitution *{Y/Z', X/Y', S/S'}*.

Absorption then replaces the body of the absorbed clause C_2 by its head properly substituted in the body of the absorbant clause C_1.

In our example, this gives the following new clause:

E_f: *grandfather(X',Z')* ←
father(X',Y') ∧ *mother(Y',Z')* ∧
tom(X') ∧ *liz(Z')* ∧ *helen(Y')*.

At this step, Absorption is not applicable anymore, remains the choice between Intraconstruction and Truncation. Let us suppose the user chooses to apply Intraconstruction.

2.2.2.2 Intraconstruction

Intraconstruction compares its two input clauses and rewrites them by introducing a new intermediary predicate to get a more concise expression of the theory. Intraconstruction can occur between two clauses C_1 and C_2 if there exists a non empty common generalization of the two clauses. This generalization must cover the heads of C_1 and C_2 and at least one literal in the bodies of C_1 and C_2. It proceeds in three steps.

Firstly, the system generates a new clause GG the head of which is the generalization of the heads of C_1 and C_2. The body of is the generalization of the bodies of C_1 and C_2. For instance, the first step of applying intraconstruction to E_f and T_1 gives the following clause:

GG: *grand-father(U,W)* ← *father(U,V)*.

The second step takes care of the left-over literals in C_1 and C_2. Here, the literals *mother(Y',Z')* ∧ *tom(X')* ∧ *liz(Z')* ∧ *helen(Y')* of E' and *father(Y,Z)* of T_1 have been left over during the generalization step of intraconstruction. Intraconstruction then introduces a new concept, arbitrarily called *newp*. This new concept is defined in two clauses whose bodies are respectively the the left-over literals in the initial clauses. The arguments of *newp* have to be carefully chosen to keep the variables bindings that were present in C_1 and C_2, such that no generalization occurs during this step. In our example, these new clauses are:

T_{1b} : *newp(Y,Z)* ← *father(Y,Z)*.
T_{1cc} : *newp(X',Y',Z')* ← *mother(Y',Z')* ∧
tom(X') ∧ *liz(Z')* ∧ *helen(Y')*.

We notice that in T_{1cc} , we need one more argument than in T_{1b} to keep to bindings between the leftover literals and the generalization. The system therefore suggests to simplify E' before proceeding to Intraconstruction in order to have symmetrical bindings for arguments of newp.

2.2.2.3 Truncation

Truncation generalizes a single clause by turning terms into variables. This is done by dropping some literals among the ones introduced by the flattening algorithm. In our example, Truncation can at most be used to turn constants of T_{1cc} into variables. Previous Intraconstruction suggested that tom(X') should be dropped from E'. It gives the more general clause:

$E_{f'''}$: *grandfather(X',Z')* ←
father(X',Y') ∧ *mother(Y',Z')* ∧
liz(Z') ∧ *helen(Y')*.

Additional information about the domain stating that first names are not relevant in this context, or oracle can allow to drop liz(Z') and helen(Y'), but we stick to $E_{f'''}$ for our example.

2.2.2.4. Intraconstruction (continued)

Going back to Intraconstruction , where the two input clauses are now T_1 and $E_{f'''}$. The generalization of the two clauses remains the same as in section 2.2.2 clauses. The two new clauses new clauses to define newp are:

T_{1b} : *newp(Y,Z)* ← *father(Y,Z)*.
T_{1cc} : *newp(Y',Z')* ← *mother(Y',Z')* ∧
liz(Z') ∧ *helen(Y')*.

The bindings for T_{1b} and T_{1cc} are now symmetrical due to the use of truncation. We can then proceed to the last step of Intraconstruction that replaces E_f and T_1 by the following three clauses:

T_{1a} : *grand-father(U,W)* ← *father(U,V)* ∧
newp(V,W).
T_{1b} : *newp(Y,Z)* ← *father(Y,Z)*.
T_{1cc} : *newp(Y',Z')* ← *mother(Y',Z')* ∧
liz(Z') ∧ *helen(Y')*.

An oracle is needed to further simplify T_{1cc} into T_{1c} using truncation :

T_{1c}: *newp(Y',Z')* ← *mother(Y',Z')*.

The oracle is then asked to validate and name the predicate *newp* . The proposed name is *parent*.

The new theory T' formed of the clauses (T_{1a}, T_{1b}, T_{1c}, T_2, T_3). This theory T' is able to recognize the new example, and all other examples of maternal grandfathers.

3 Saturation

3.1 Two problems with inverse resolution

While implementing the above described operators in the Ires system, we have discovered some rather fundamental limitations of inverse resolution.

3.1.1 Truncation

The first problem when we consider non unit clauses arises with Truncation. Truncation as pure inversion of resolution operator allows to get rid of literals introduced during the flattening step. It corresponds in a functional[2] notation to generalize a clause by replacing some terms

[2] By functional notation we mean the unflattened representation for clause, as opposed to the flattened representation.

by variables. It seems appealing to extend Truncation so that we can drop *any* literal in a flattened representation. Dropping literals that were not introduced by flattening corresponds to the dropping condition rule in a functional representation (Michalski, 1983). One single rule in a flattened representation (dropping any literal) is sufficient to express the two generalization rules that are necessary in a functional representation (that is, turning terms into variables, dropping literals).

The problem with this extension is that it is out of the scope of pure inversion of resolution. From a clause $A \leftarrow B \wedge C$, we can now obtain the clause $A \leftarrow C$ with Truncation. However this is not the reverse of a resolution step. Using SLD resolution it is not possible to derive the clause $A \leftarrow B \wedge C$ from the clause $A \leftarrow C$.

This is one of the reasons that led us to introduce a different definition for generality. A is more general than B if $A \models B$. This definition allows us to say $A \leftarrow C$ is more general than $A \leftarrow B \wedge C$ because

$(A \leftarrow C) \models (A \leftarrow B \wedge C)$,

whereas

$(A \leftarrow C) \not\vdash_{SLD} (A \leftarrow B \wedge C)$.

This extension of our definition of generality will be also justified by the sections to come.

3.1.2 Absorption

Saturation has its origin in some chaining problem we had with Absorption. Absorption is destructive : it replaces in the body of one clause the body of another clause by its head. The literals that have been replaced are lost, and this can have harmful effects on further generalizations or Intraconstructions if the wrong choice has been made for the clauses involved in the Absorption, or concerning the order in which Absorptions occurs[3]. For the sake of clarity, let us take an example with the two following clauses in our domain theory :

$A \leftarrow B \wedge C \wedge E$

$B \leftarrow C \wedge D$.

and then the following example comes

$F \leftarrow C \wedge D \wedge E$.

Absorption of the example with the second clause would yield the clause

$F \leftarrow B \wedge E$

Literals C and D are removed, thus preventing one more absorption with the first clause of the theory. Beside the fact that Absorption is not complete since all possibilities cannot be tried, this have another bad effect. The order into which Absorption is applied is very important. However the good choice cannot be known in advance. Thus a large search has to be performed.

A better solution would be to keep all the literals in the body of the clause when performing Absorption. We call this method **Saturation.** This would first yield the clause

$F \leftarrow [C \wedge D] \wedge E \wedge B$

then applying Absorption again with the first clause gives

$F \leftarrow [C \wedge D \wedge E \wedge B] \wedge A$

Brackets denote that the literals have been already used for an absorption step, and are therefore optional (they can be dropped in a later simplification step). This notation is taken from (Loveland, 1978).

Thus the Saturation operator overcomes the destructive effects of Absorption by doing a kind of forward chaining on the body of one clause as the following logical analysis will show. It also offer a good alternative to the search needed for Absorption. The clause

$F \leftarrow [C \wedge D \wedge E \wedge B] \wedge A$

is a compact way of representing all the clauses that should to be tried with Absorption. These clauses are obtained by removing some of the bracketed literals. Some of the possible 16 clauses are given below :

$F \leftarrow C \wedge D \wedge E \wedge B \wedge A$

$F \leftarrow D \wedge E \wedge B \wedge A$

$F \leftarrow E \wedge A$

$F \leftarrow A$

3.2 Logical analysis

The preceding remarks led us to introduce a sensibly different framework for induction. The starting hypothesis remains the same: a domain theory T (definite clause program) is given, as well as an example clause ($B \leftarrow B_1 \wedge B_2 \wedge .... \wedge B_n$) not explained by the domain theory, i.e. not entailed by T. The goal of learning is to obtain a clause H, known as induction hypothesis, such that

$$T \wedge H \models (B \leftarrow B_1 \wedge B_2 \wedge .... \wedge B_n)$$

This can be transformed into

$$T \wedge \neg (B \leftarrow B_1 \wedge B_2 \wedge .... \wedge B_n) \models \neg H$$

The next step is to remove all the variables in the negated example clauses by skolemizing them. Skolemizing amounts to apply a substitution θ involving constants which do not appear in the theory.

$$T \wedge \neg (B \leftarrow B_1 \wedge B_2 \wedge .... \wedge B_n)\theta \models \neg H$$

This can be transformed into

$$T \wedge \neg B\theta \wedge B_1\theta \wedge B_2\theta \wedge .... \wedge B_n\theta \models \neg H$$

The problem of finding H is now reduced to deduction. Our solution is to use a forward chaining deduction on $B_1\theta \wedge B_2\theta \wedge .... \wedge B_n\theta$ (which is the body of the example clause) using the theory T. Let us consider any conjunction of atoms $A_1 \wedge A_2 \wedge .... \wedge A_m$ that can be proved that way. We thus have

$$T \wedge B_1\theta \wedge B_2\theta \wedge .... \wedge B_n\theta \models A_1 \wedge A_2 \wedge . ... \wedge A_m$$

We then have, by adding $\neg B\theta$ to both sides

$$T \wedge \neg B\theta \wedge B_1\theta \wedge B_2\theta \wedge .... \wedge B_n\theta \models \neg B\theta \wedge A_1 \wedge A_2 \wedge .... \wedge A_m$$

By factorizing negations we obtain

$$T \wedge \neg (B \leftarrow B_1 \wedge B_2 \wedge .... \wedge B_n)\theta \models \neg (B\theta \leftarrow A_1 \wedge A_2 \wedge .... \wedge A_m)$$

[3] This point has been made by S; Muggleton in a private communication.

The last step is to undo the skolem substitution, i.e. replace each skolem constant introduced by θ with a new variable. We note this operation θ^{-1}. We have proved that we then have the logical entailment :

$$T \wedge \neg (B \leftarrow B_1 \wedge B_2 \wedge .... \wedge B_n) \models \neg (B\theta \leftarrow A_1 \wedge A_2 \wedge .... \wedge A_m)\theta^{-1}$$

Thus H is taken as $(B\theta \leftarrow A_1 \wedge A_2 \wedge .... \wedge A_m)\theta^{-1}$. We have also proved that all the hypothesis H having the same predicate symbol in the head as in the example can be obtained that way, which was not the case with Absorption.

The last refinement of this method is the use of optional literals (which are put between brackets) when performing deduction . This refinement of resolution is analyzed in details in (Loveland, 1978).

Saturation has already been applied in the context of generalization (Bisson, 1989), although the use of saturation to get to a generalization in the two approaches is quite different.

3.3 Example

We present the principle of Saturation on an example taken from (Wirth, 1988) as the application domain, namely completion of Definite Clause Grammars, is particularly suited for this. The domain theory expresses natural language parsing rules. Casting Wirth's representation into ours gives the following :

(LFP1): sentence(S0,S) ←
noun_phrase(S0,S1) ∧
verb_phrase(S1,S)
(LFP2) : noun_phrase(S0,S) ←
determiner(S0,S1) ∧
noun(S1,S).
(LFP3): noun_phrase(S0,S) ← name(S0,S).
(LFP4): verb_phrase(S0,S) ←
intransitive_verb(S0,S).
(LFP5): noun(X,S) ← $cons_p$(man,S,X).
(LFP6): name(X,S) ← $cons_p$(sue,S,X).
(LFP7): determiner(X,S) ← $cons_p$(a,S,X).
(LFP8): determiner(X,S) ← $cons_p$(the,S,X).
(LFP9): intransitive_verb(X,S) ← $cons_p$(sleeps,S],X).
(LFP10): transitive_verb(X,S) ← $cons_p$(loves,S,X).

We start with the flat representation of the input clause *sentence([sue,loves,a,man/S],S)* (for the sake of readability, we did not flatten constants in the example) :

(I): sentence(S0,S) ←
$cons_p$(sue,S1,S0) ∧ $cons_p$(loves,S2,S1) ∧
$cons_p$(a,S3,S2) ∧ $cons_p$(man,S,S3).

This example is skolemized by replacing all variables by new constants s, s1,s2,s3:

(Is) : sentence(s0,s) ←
$cons_p$(sue,s1,s0) ∧ $cons_p$(loves,s2,s1) ∧
$cons_p$(a,s3,s2) ∧ $cons_p$(man,s,s3).

Saturation builds in one pass the bottom up parsing of this sentence, rewriting parts of the clauses in terms of higher level concepts. The result of Saturation will be :

(Iss) : sentence(s0,s) ←
[$cons_p$(sue,s1,s0) ∧ $cons_p$(loves,s2,s1) ∧
$cons_p$(a,s3,s2) ∧ $cons_p$(man,s,s3) ∧
name(s0,s1)] ∧ determiner(s2,s3) ∧ noun(s3,s)
∧ transitive_verb(s1,s2) ∧ noun_phrase(s0,s1)
∧ noun_phrase(s2,s).

The last step is to undo the skolem substitution, which gives :

(If) : sentence(S0,S) ←
[$cons_p$(sue,S1,S0) ∧ $cons_p$(loves,S2,S1) ∧
$cons_p$(a,S3,S2) ∧ $cons_p$(man,S,S3) ∧
name(S0,S1)] ∧ transitive_verb(S1,S2) ∧
[determiner(S2,S3) ∧ noun(S3,S)] ∧
noun_phrase(S0,S1) ∧ noun_phrase(S2,S).

From now on we will not indicate the skolemizing step since it is straightforward. The result of Saturation can be clearly seen on the following graph, where nodes stand for literals of the clause and arrows denotes the deduction made while saturing. Low level as well as high level concepts appears in the body of the clauses. There are of course redundancies, but all the information contained in the body of the clause is explicit.

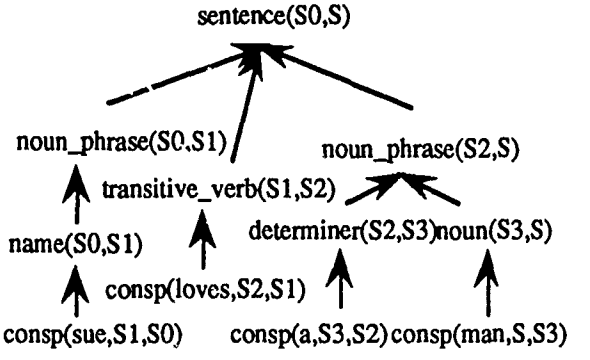


Figure1 : Graphical representation of a saturated clause

3.4 Boundaries of generality

The above graph can be interpreted with a generality relation : the higher level literals are more general than the others. This seems to contradict the definition of generality we have used before since the higher literals are deduced from the lower ones.

In fact, what is compared is not literals, but clauses. In fact the above graph potentially represents a lot of different clauses, and what should be compared for generality are these clauses, not simple literals. To obtain one of these clauses, one should draw a line across the graph, called **boundary of generality**, and keep all the literal just above this line. For instance, if we use the

boundary of generality (B1) in figure 2, we obtain the clause :

sentence(S0,S)←
noun_phrase(S0,S1) ∧ transitive-verb(S1,S2) ∧ noun_phrase(S2,S3).

If take the boundary of generality (B2) we obtain :
sentence(S0,S)←
$cons_p$(sue,S1,S0) ∧ transitive-verb(S1,S2) ∧ noun_phrase(S2,S3).

The first clause is more general than the second one. This is why we can say that *noun_phrase(S0,S1)* is more general than *$cons_p$(sue,S1,S0)* with respect to the above graph. Thus the graph represents generality level of literals : the higher a literal is, the more general will be a clause with this literal in the body. Note that this notion of boundary of generality is similar to the one opf boundary of operationality from (Braverman & Russel, 1988). This will be used in the intraconstruction and truncation operators as described in section 4.2.

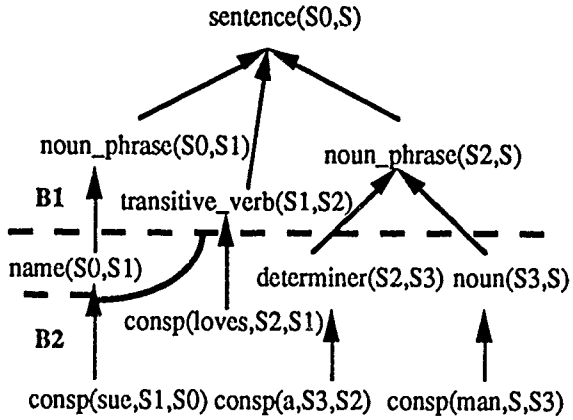


Figure 2 : Saturated clause with boundaries of generality

Depending on the learning goal, it is now possible to favour higher level terms, and to prune some parts of this graph. The control of the operators is thus easier than with Absorption : all the relevant information is explicit when choices have to be made. Various bias can be used at this point. If we retain only the connected top (see section 4 below) we obtain the clause :
(Ir) : sentence(S0,S)←
noun_phrase(S0,S1) ∧ transitive-verb(S1,S2) ∧ noun_phrase(S2,S3).
which was the missing clause in the theory. This clause can now undergo intraconstruction with (LFP1) to complete the definition of *verb_phrase*.

4. Intraconstruction and Truncation revisited

4.1 General algorithm of N-IRES

The previous remarks implies a total rethinking of our previous approach to inversion of resolution. Intraconstruction and Truncation will from now on operate on the graphs produced by Saturation. This will allow us to define explicit control by taking into account level of generality in the graphs. The generalization step of Intraconstruction becomes a matching between the two input graphs, where nodes are literals partially ordered with subsumption. As the literals do not have the same generality level, Truncation operates with higher priority on low level literals, which are already subsumed by some other terms in the generalization. Truncation becomes a central operator that prevents the resulting clauses after Saturation to grow immoderately in size. Intraconstruction also benefits form the graph structure. Of course, partial matching of two graphs is np-complete, but matching can be seen as finding some paths in the graph starting from the heads of the clauses (we are then brought back to a tree-matching problem). We can use as well the fact that the literals are ordered in the graph to constrain the order in which the literals are going to be generalized. A possible bias for the generalization step is proposed in the next section.

The global algorithm of N-IRES is now the following. We suppose that examples comes one by one, and are handled as soon by the system.

For each new example, a subsumption test is performed (like the one from (Buntine, 1988)). If the example is subsumed by the domain theory, nothing is done. If the example is not explained by the theory, the input is saturated using the original domain theory, then it may undergo Intraconstruction or Truncation. The results of these operators are submitted to the oracle that validates them or not. Once a result has been validated, it is added to the domain theory. The system then checks whether any clause of the original theory can be simplified (by a sequence of Saturation - Intraconstruction - Truncation) using this new clause. If it is the case, the expression of the theory is simplified. This is similar to the rule-reexamination from (Hall, 1988) and allow the system to be less sensitive to the order of the examples.

The small following example gives a hint of improvements of Intraconstruction made possible by Saturation.

4.2 Example

We concentrate here on learning the rule for the general case of arithmetic addition. Integers are represented using the constant 0 and the successor function s: s(X) represents the integer X+1.

The first input is
(I1) : 1+1 = 2, which is represented by *plus(s(0),s(0),s(s(0)))*.
Since two identical constants are replaced by identical terms, this will give the following flattened representation :
(I1f) : plus(X,X,Y) ←
zero(ζ) ∧ *succ(ζ,X)* ∧ *succ(X,Y)*.
The next input is (I2) : 1 + 2 = 3. Its flattened representation is
(I2f) : plus(X,Y,Z) ←
zero(ζ) ∧ *succ(ζ,X)* ∧ *succ(X,Y)* ∧ *succ(Y,Z)*.
Saturation is fired, giving the clause :
(I2fs) : plus(X,Y,Z) ←
zero(ζ) ∧ *succ(ζ,X)* ∧ *succ(X,Y)* ∧ *succ(Y,Z)* ∧ *plus(X,X,Y)*.
This can be described graphically as follows, using the same conventions as in figure 1:

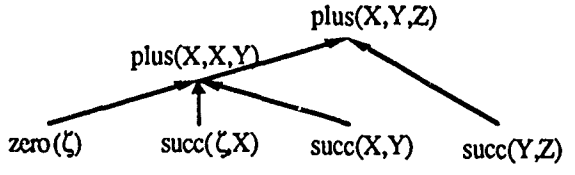


Figure 3: Saturated graph of '1+2 = 3'

The third example is (I3) : 1 + 3 = 4. The corresponding flattened clause is :
(I3f) : plus(X',Z',U') ←
zero(ζ') ∧ *succ(ζ',X')* ∧ *succ(X',Y')* ∧ *succ(Y',Z')* ∧ *succ(Z', U')*.
Saturation provides (firing I1fs and I2fs on the body of I3f) the clause :
(I3fs) : plus(X',Z',U') ←
zero(z') ∧ *succ(ζ',X')* ∧ *succ(X',Y')* ∧ *succ(Y',Z')* ∧ *succ(Z', U')* ∧ *plus(X',X',Y')* ∧ *plus(X',Y',Z')*.

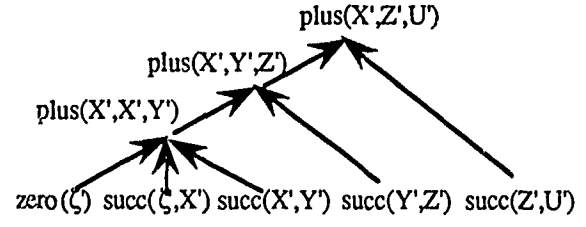


Figure 4 : Saturated graph of '1+3 =4'

We then compare the two graphs that represent the saturated clauses. They are compared because their two heads match (which was not the case for I1f and I2f) in order to find some regularities in the expression of the two additions. The first step is to propose the generalization produced by Truncation to the user. If the user does not validate the generalization on its own, N-IRES then proceeds to Intraconstruction (i.e., introduces a new predicate to cover the leftover literals), where no induction is done.

Generalizing means finding a partial match of the two graphs. Finding the maximal partial match is np-hard, but on this example, we are going to show that we are not necessarily interested in the maximal partial match. The heads of the clauses are first generalized, and then the match is extended by exploring in priority nodes which are near the root node (most general generalizations are privileged).

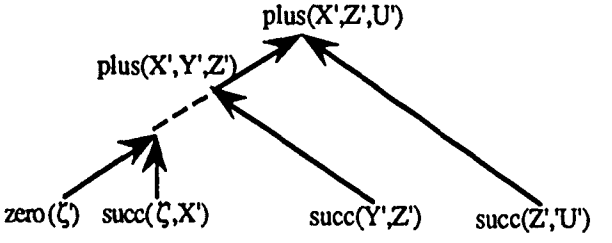


Figure 5 : Best graph matching for learning addition

The generalization is :
(G) : plus(A,B,C) ←
plus(A,D,B) ∧ *succ(B,C)* ∧ *succ(D,B)* ∧ *zero (ζ'')* ∧ *succ(ζ'',A)*.
This generalization covers completely the I2fs graph, and leaves two nodes of the I3fs graph unmatched. It is near from the expression we wish to learn, but it still contains irrelevant terms.

4.3 Connectivity

This generalization can be further simplified : literals that are already covered by some terms in the generalization, (such as *zero(ζ)* and *succ(ζ,A))* can be dropped from the generalization, as long as it remains connected (roughly, there must exist one path linking each variable in the body of the generalization to one variable that appears in the head predicate through predicates in the clause). This can be called the *connectivity heuristic* and it keeps us with the minimal links (one path) between variables in the body of the generalization and variables in its head).

We are looking for relaxation of this fairly strong contraint. A similar kind of constraint has been considered in (Helft , 1988). Using simplifying heuristics should lead us to drop the two literals *zero(ζ)* and *succ(ζ,A)*. We thus obtain :
plus(A,B,C) ←
plus(A,D,B) ∧ *succ(B,C)* ∧ *succ(D,B)*
which gives once unflattened:

plus(A,succ(D),succ(B)) ← plus(A,D,B). This means that A+succ(D) = succ (A+D). We recognize here a definition of addition in the general case.

5 Conclusion

After implementing a simple solution to inversion of resolution when the representation language used does not include function symbols in the IRES system, we noticed some more fundamental limitations to the inversion of resolution approach. The major limit affects the Absorption operator. To overcome this limit, we relate our work to results in Logic Programming and replace the Absorption operator with the Saturation operator that makes all the possible deductions on the body of one clause. To handle efficiently saturated clauses, we represent them in the form of a graph ordered with the subsumption relation. Intraconstruction and Truncation then operate on these new structures that allow to reduce search for applicable operator. In the same way, the graph structure allow to easily express bias to constrain the search for a good generalization while applying Saturation and Intraconstruction (details can be found in (Rouveirol,1990)).

We see two major research directions at this point. First of all, we are going to study way to express bias to simplify the graphs after Saturation (that is, how to choose among all the potential generalizations contained in a saturated clause and how to prune the irrelevant terms) and to constrain the graph matching step in Intreaconstruction. A second important issue will be to evaluate the system, and in particular to scale up to large applications.

Acknowledgements: We would like to thank Yves Kodratoff, who is our thesis supervisor for the support he gave to this work, Steve Muggleton and Nicolas Helft for interesting discussions, and all the members of the Inference and Learning group at LRI.

References

Bisson G.: "APOGEE : une utilisation de la saturation pour généraliser", in *Actes des quatrièmes Journées Françaises de l'Apprentissage*, pp31-45,1989.

Braverman M. S. & Russel S. J. : "Boundaries of operationality", in *proceddings of the fifth International Conference on Machine Learning*, Ann Arbor, pp 221-235, Morgan Kaufman, 1988.

Buntine W.: "Generalised subsumption and its applications to redundancy", *Artificial Intelligence*, 36, 149-176, 1988

Genesereth M. R., Nilson N. J., *Logical Foundations of Artificial Intelligence*, Morgan Kaufman, 1987.

Hall R. J.: "Learning by failing to explain", *Machine Learning Journal, vol 3-1*, pp 45-77, august 1988.

Helft N. : "L'induction en Intelligence Artificielle : Théorie et Algorithmes", *thèse de l'Universté d'Aix-Marseille II*, septembre 1988.

Kodratoff Y. & Ganascia J.G.G. : "Improving the generalisation step in Learning", in *Machine Learning, An Artificial Intelligence Approach*, volume II, pp 215-244, Morgan Kaufman, 1986.

Kodratoff Y. : *Introduction to Machine Learning*, Pitman, 1988.

Lloyd J.W.: *Foundations of Logic Programming*, second extended edition, Springer Verlag, 1987.

Loveland D. W.: *Automated Theorem Proving : A Logical Basis*, North Holland, 1978

Michalski R.S. : A Theory and a Methodology of Inductive Learning, *Artificial Intelligence, vol 20*, pp 111-161, 1983.

Mitchell T. M.: "Generalization as search", Artificial Intelligence 18, pp203-226, 1982

Muggleton S. & Buntine W.: "Machine invention of first order predicates by inverting resolution", proceedings of 5th international Machine Learning Workshop, Morgan Kaufman, pp 339-352

Muggleton S. : "A strategy for constructing new predicates in first order logic", *proceedings of EWSL 88*, Pitman, pp 123-130.

Niblett T.: "A study of generalisation in logic programs", *proceedings of EWSL 88*, Pitman, p 131-138.

Plotkin : "A further note on inductive generalisation", in Meltzer and Michie editors, *Machine Intelligence 6*, pp 101-124, Edimburgh University Press

Sammut C.: "Learning Concepts by Performing Experiments", PhD dissertation, University of New South Wales, Kingston , 1981.

Rouveirol C. & Puget J.F.: "A simple solution for Inverting Resolution", in *Proceedings of the 4th European Working Session on Learning*, pp201-211, Pitman, 1989.

Rouveirol C. : "Saturation : Postponing choices when Inverting Resolution", proceedings of ECAI90, to appear, 1990.

Wirth R.: "Learning by Failing to Prove", *proceedings of the third European Working Session on LEarning*, Pitman, pp 237-251,1988.

GENETIC ALGORITHMS

GENETIC PROGRAMMING

Building Artificial Nervous Systems Using Genetically Programmed Neural Network Modules

Hugo de Garis

CADEPS Artificial Intelligence Research Unit,
Universite Libre de Bruxelles (U.L.B.),
Ave F.D. Roosevelt 50, C.P. 194/7, B-1050, Brussels, Belgium.
tel: + 32 2 642 2783, email: CADEPS@BBRNSF11(.BITNET)

Keywords

Genetic Algorithm, Time Dependent Neural Network Modules, GenNets, Genetic Programming, Artificial Nervous Systems, Sequential Evolution, Behavioural Memory, Brain Building, Nanobrain, Darwin Machines.

Abstract

This paper introduces a new programming methodology, called Genetic Programming, which is the application of the Genetic Algorithm to the evolution of the signs and weights of fully (self) connected neural network modules which perform some time (in)dependent function (e.g. walking, oscillating etc.) in an "optimal" manner. Genetically Programmed Neural Net (GenNet) modules are of two types, functional and control. A series of functional GenNets can be evolved, and their weights frozen. Control GenNets are then evolved whose outputs are the inputs of the functional GenNets. The size and timing of these control signals are evolved such that the combination of control and functional GenNets performs as desired. This combination can then be frozen and used as a module in a more complex structure. This procedure can be repeated indefinitely, thus allowing the construction of hierarchical neural networks. Genetic Programming has recently proven to be so successful that the building of artificial nervous systems becomes a real possibility.

This paper illustrates both the conceptual simplicity and the power of Genetic Programming by showing how a GenNet can be evolved which teaches a pair of stick legs to walk. This is followed by a description of work in progress on the next major phase of Genetic Programming research, namely the building of artificial nervous systems ("brain building"), and on the tools which will be needed to evolve them, called Darwin Machines.

Introduction

This paper shows that fully (self) connected neural networks [RUMELHART et al 1986], whose weights and signs are evolved with the Genetic Algorithm [GOLDBERG 1989], are powerful enough to control a system which is highly time dependent in its behaviour, e.g. a pair of walking stick legs. Normally, recurrent neural networks (i.e. those with feedback links between neurons) require a certain "settling time" for the output neurons to stabilize their output signal values given fixed (clamped) input values to the input neurons.

What is interesting in the experiments in this paper is that the inputs change faster than the settling time, so that the neural network never settles. Under such circumstances, it is not obvious how the usual neural network learning algorithms can be applied to teach such a network to walk. The Genetic Algorithm (GA) however does not really care how the neural network performs this task, so long as it performs it.

Section 1 of this paper summarises briefly the principles of the Genetic Algorithm (GA). Section 2 introduces the concept of Genetic Programming. Section 3 describes the particular GenNet used and how the GA is applied to its evolution. Section 4 presents the results of some GenNet experiments. Section 5 discusses ideas for future research and in particular, the concepts of Brain Building and the Darwin Machine.

1 The Genetic Algorithm

The Genetic Algorithm is a form of simulated evolution to solve optimization problems in a Darwinian

"survival of the fittest" approach. Solutions to problems are coded onto (usually binary) strings called "chromosomes", (e.g. parameter values in a control problem), which compete with each other to reproduce the next generation. A quality value for the encoded solution of each chromosome is determined, and the probability of reproduction of each chromosome into the next generation is proportional to this value. The number of chromosomes per generation remains fixed. Genetic operators such as mutation (bit flipping), crossover (cutting two chromosomes at the same position and swapping portions), inversion (inverting a section of a chromosome). Occasionally, the application of these operators to the offspring causes them to have higher quality values than their parents. Hence they will reproduce with higher probability, and squeeze out inferior chromosomes. Over time, the average quality of the population will increase. The GA can be seen as a form of hill climbing where there may be many hills in the configuration space. For an excellent introduction to the principles of Genetic Algorithms see [Goldberg 1989].

2 Genetic Programming

Genetic Programming is a new programming methodology which uses the Genetic Algorithm to design neural network modules [de GARIS 1990]. The programmer specifies the behavioural characteristics that the neural network should possess, the number of neurons in the network (which will be fully (self) connected), identifies the input and the output neurons, the quality criterion for the performance of the network, the number of binary places after the binary point of the numbers which represent the size of the weights connecting neurons, the initial input signal values, the number of cycles, etc. The user also provides a list of parameters necessary to control the GA, such as the number of generations, the size of the population, the probability of mutation, the size of the scaling factor to avoid premature convergence, etc. A concrete example is shown in the next two sections.

The GA is then used to find both the signs and the values of the weights of the network which provides the functionality desired. Once these weights are found, they are frozen (fixed) and the GenNet module thus evolved can be used as a component in a more complex structure. Usually a set of GenNet modules consists of a subset of low level modules which execute some simple functions. These low level modules are usually managed by control modules which are themselves GenNets, i.e. they too are evolved with the GA. The outputs of the control modules are the inputs to the low level modules (without any intervening weights). Once the control and low level modules (considered now as a unit) are functioning together as desired, the weights of the control modules are frozen and the unit can then be considered as a module or component for an even more complex structure.

If one can analyze a complex behaviour (of a system) into its component behaviours and their corresponding control inputs, then Genetic Programming is a useful approach. One can solve problems in this way which are too complex to solve with traditional algorithmic approaches.

3 The GenNet

One of the aims of this paper, as mentioned in the abstract, is to show that GenNets are powerful enough to provide highly time dependent control. The vehicle used to show this possibility is a simple pair of stick legs, which is to be taught to walk. FIG. 1 shows the basic setup. The output values of the GenNet are interpreted to be the angular accelerations of the four components of the legs.

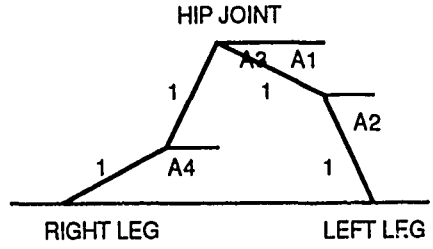


FIG. 1

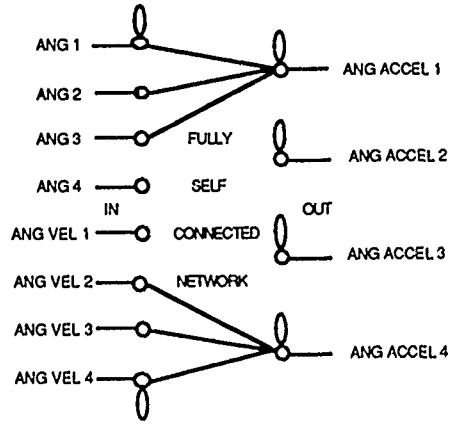


FIG. 2

Knowing the values of the angular accelerations (assumed constant over one cycle - where a cycle is the time period over which the neurons calculate (synchronously) their outputs from their inputs), and knowing the values of the angles and the angular velocities at the beginning of the cycle, one can calculate the values of the angles and the angular velocities at the end of the cycle.

As input to the GenNet (control module) were chosen the angles and the angular velocities. FIG.2 shows how this feedback works. Knowing the angles, one can readily calculate the positions of the two "feet" of the stick legs. Whenever one of the feet becomes lower than the other, that foot is said to be "on the ground", and the distance (whether positive or negative) between the positions of the newly grounded foot and the previously grounded foot is calculated.

The aim of the exercise is to evolve GenNets which make the stick legs move as far as possible to the right in the user specified number of cycles, and cycle time. The GenNet used here consists of 12 neurons; 8 input neurons and 4 output neurons (no hidden neurons). The 8 input neurons have as inputs the values of the 4 angles and the 4 angular velocities. The input angles range from -1 to +1, where +1 means one half turn (i.e. 180 degrees). The initial (start) angles are chosen randomly, ranging between 0 and 1. The initial angular velocities are chosen randomly between -1 to +1, and are given in half turns per secon

The activity of a neuron is calculated in the usual way, namely the sum of the products of its inputs and its weights, where weight values range from -1 to +1. The output of a neuron is calculated from this sum, using the symmetrical sigmoid function
(-1 + (2/(1 + exp(-sum)))), which ranges fro -1 to +1. The outputs of neurons are restricted to have absolute values of less than 1 so as to avoid the risk of explosive positive feedback.

The chromosomes used to evolve the weights and their signs in the GA are simple binary strings. The user specifies the number P of binary places after the binary point of the numbers representing the values of the weights (where weights have an absolute value less than 1). Imagine this is 6. One bit is used per weight to specify the sign of the weight (0 is positive, i.e. an excitory "synapse", 1 is negative, i.e. an inhibitory "synapse"). Thus, for a GenNet of N neurons, a chromosome will be N*N*(P + 1) bits long, since there are N*N weights in a fully (self) connected network.

The initial population of chromosomes is generated randomly with each bit equally likely to be a 0 or a 1. The ith group of (P + 1) bits corresponds to the sign bit followed by the P bits giving the weight value of the ith weight, e.g. 100101 represents a weight of -0.15625.

For our experiments, no crossover was used. GenNet neurons are so highly interdependent that crossing over chromosome portions is detrimental. GenNet GAs typically function with no sex (i.e. no crossover) and no inversion, using only mutation and selection. The selection technique used was standard "roulette wheel", (see [GOLDBERG 1989]).

The quality criterion used for selecting the next generation was usually the total distance covered by the stick legs in the total time T, where (T = C*cycletime) for the user specified number C of cycles and cycletime. The quality is thus the velocity of the stick legs moving to the right. Right distances are non negative. Stick legs which moved to the left scored zero and were eliminated after the first generation.

4 Results

A series of experiments was undertaken. In the first experiment, no constraint was imposed on the motion of the stick legs (except for a selection to move right across the screen). The resulting motion was most un-lifelike. It consisted of a curious mixture of windmilling of the legs and strange contortions of the hip and knee joints. However it certainly moved well to the right, starting at random angles and angular velocities. As the distance covered increased, the speed of the motion increased as well and became more "efficient", e.g. windmilling was squashed to a "swimmers stroke".

In the second experiment, the stick legs had to move such that the hip joint remained above the floor (a line drawn on the screen). During the evolution, if the hip joint did hit the floor, evolution ceased, the total distance covered was frozen, and no further cycles were executed. After every cycle, the coordinates of the two feet were calculated and a check made to see if the hip joint did not lie below both feet. This time the evolution was slower, presumably because it was harder to find new weights which led to a motion satisfying the constraints. The resulting motion was almost as un-lifelike as in the first experime ., again with windmilling and contortions, but at least the hip joint remained above the floor.

In the third experiment, a full set of constraints was imposed to ensure a lifelike walking motion. The result was that the stick legs moved so as to take as long a single step as possible, and "did the splits" with the two legs as extended as possible and th. .. nt just above the floor. From this position, it wa ss ble to move any further. Evolution ceased. T luable lesson and focussed attention upon the t concept of "evolvability", i.e. the capacity for further evolution. A change in approach was needed.

This led to the concept of Behavioural Memory, i.e. the tendency of a behaviour which was evolved in an earlier phase to persist in a later phase. For example, in phase 1, evolve a GenNet for behaviour A. Take the weights and signs resulting from this evolution as initial values for a second phase which evolves behaviour B. One notices that behaviour B contains a vestige of behaviour A. This form of Sequential Evolution can be very useful in Genetic Programming, and was used to teach the stick legs to walk, i.e. to move to the right in a step like manner, in a 3 evolutionary phases.

In the first phase, a GenNet was evolved over a short time period (e.g. 100 cycles) which took the stick legs from an initial configuration of left leg in front of right leg to the reverse configuration. The angular velocities were all zero at the start and at the finish. By then allowing the resulting GenNet to run for a longer time period (e.g. 200 cycles) the resulting motion was "step-like", i.e. one foot moved in front of and behind the foot

on the floor, but did not touch the floor. The quality measure for this GenNet was the inverse of the sum of the squares of the differences between the desired and the actual output vector components (treating the final values as an 8 component state vector of angles and angular velocities).

This GenNet was then taken as input to a second (sequential evolutionary) phase which was to get the stick legs to take short steps. The quality measure this time was the product of the number of net positive steps taken to the right, and the distance. The result was a definite stepping motion to the right but distance covered was very small. The resulting GenNet was used in a third phase in which the quality measure was simply the distance covered. This time the motion was not only a very definite stepping motion, but the strides taken were long. The stick legs were walking.

The above experiments were all performed in tens to a few hundred generations of the GA. Typical GA parameter values were - population size = 50; mutation rate = 0.001; scaling factor = 2.0 (a linear scaling of quality scores in the GA (max.value = sc. fact.*av.value), which prevents premature convergence); cycletime = 0.03; number of cycles = 200 to 400.

A video of the results of the above experiments has been made. To obtain a real feel for the evolution of the motion of the stick legs, one really needs to see it.

5 Future Research - Brain Building and the Darwin Machine

The great advantage of Genetic Programming in comparison to traditional neural network learning schemes (e.g. the popular backpropagation or backprop method [RUMELHART et al 1986]) is that it can be unsupervised, i.e. one need not know the desired output vectors of the neural network. If one is dealing with a time dependent phenomenon in a supervised method, one needs to know the desired output vectors as a function of time, which implies that one already understands the phenomenon well enough to be able to know what the outputs should be. With Genetic Programming, the only thing that is necessary is that one has a quality measure (usually a scalar) of the performance as a whole, which can be fed back to the device being Genetically Programmed.

The success of these experiments raises the fascinating prospect of building artificial nervous systems ("brain building"), i.e. combining functional and control GenNets to build simple brains.

Obviously, if one can evolve one GenNet, one can evolve many, each having its own characteristic properties and behaviour. By combining the actions of many GenNets, it seems likely that it will be possible to build artificial nervous systems or "nanobrains". The term "nanobrain" is appropriate, considering that the human brain contains some trillion neurons, and that a nanobrain would therefore contain the order of hundreds of neurons. Simulating several hundred neurons is roughly the limit of present day technology, but this will change. A discussion of future prospects in this regard will follow later.

This section is concerned principally with the description of the simulation of just such a nanobrain. The point of the exercise is to show that this kind of thing can be done, and if it can be done successfully with a mere dozen or so GenNets as building blocks, it will later be possible to design artificial nervous systems with GenNets numbering in the hundreds, thousands and up. One can imagine in the near future, whole teams of human Genetic Programmers devoted to the task of building quite sophisticated nano (micro?) brains, capable of an elaborate behaviour range.

To show that this is no pipe dream, a concrete proposal will now be presented showing how GenNets can be combined to form a functioning (simulated) artificial nervous system. The implementation of this proposal has not yet been completed (at the time of writing), but progress so far inspires confidence that the project will be completed successfully. The "vehicle" chosen to illustrate this endeavour is shown in FIG. 3.

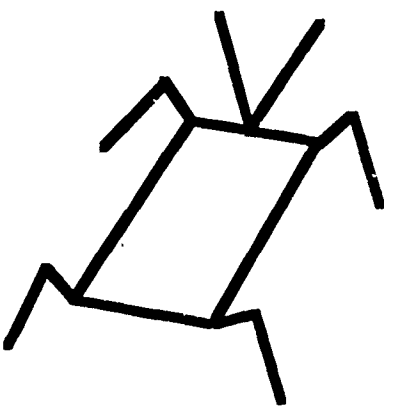


FIG. 3

This lizard-like creature (called LIZZY) consists of a rectangular wire-frame body, four two-part legs and a fixed antenna in the form of a V. LIZZY is capable of reacting to three kinds of creature in its environment, namely :- mates, predators and prey. These three categories are represented by appropriate symbols on the simulator screen. Each category emits a sinusoidal signal of a characteristic frequency. The amplitudes of all these signals decrease inversely as a function of distance. Prey emit a high frequency, mates a middle frequency, and predators a low frequency.

The antenna picks up the signal continuously. Once the signal strength becomes large enough (a value called the "attention" threshold), LIZZY detects the frequency of this signal, and depending upon the outcome, executes an appropriate sequence of actions. If the object is a prey, LIZZY rotates toward it, moves in the direction of the object until the signal strength is a maximum, stops,

alligns its legs, pecks at the prey like a hen (by pushing the front of its body up and down with its front legs), and after a while, moves away randomly. If the object is a predator, LIZZY rotates away from it, and flees until the signal strength is below the attention threshold. If the object is a mate, LIZZY rotates toward it, moves in the direction of the object until the signal strength is a maximum, stops, alligns its legs, mates (by pushing the back of its body up and down with its back legs), and after a while, moves away randomly.

The above is merely a sketch of LIZZY's behavioural repertoire. In order to allow LIZZY to execute these behaviours, a detailed circuit of GenNets and their connections needs to be designed. FIG. 4 shows an initial attempt at designing such a circuit. The black dots indicate a blocker function, i.e. if the signal is active (non zero), then the traversing signal is blocked. There are 7 different motion GenNets, of which "random move" is the default option. When any of the other 6 motions is switched on, the random move is switched off by a "blocker". LIZZY moves in the following way. Each motion GenNet consists of 8 output neurons, whose output values (as in the WALKER GenNet) are interpreted as angular acceleratations (rather than as angles, which would pose continuity problems when switching from one motion GenNet to another). The position of the upper (and lower) part of each leg is defined by the angle that the leg line takes on a cone whose axis of symmetry is equidistant from each of the XYZ axes defined by the body frame.The leg line is confined to rotate around the surface of this cone. This approach was chosen so as to limit the number of degrees of freedom. If each leg part had two instead of one degree of freedom, and one wanted to keep the GenNet fully connected, with 16 outputs and 32 inputs (angles, and angular velocities), i.e. 48 neurons per GenNet, hence 48 squared connections, the resulting chromosome would have been huge.

LIZZY's motion is determined as follows. One calculates the positions of the legs relative to the body frame. LIZZY as a whole is raised (or lowered) vertically so that the lowest "foot" touches the "floor" at a point called the "base point". LIZZY is then rotated about the base point in the vertical plane cutting the base point and the next lowest "foot" until this second foot touches the floor at the second base point. LIZZY is then rotated about the line joining the two base points until the third lowest foot touches the floor. The fourth foot will normally be raised relative to the plane defined by the first three feet.When one motion GenNet is switched off and another switched on, the current angles and angular velocities are input to the new GenNet, to ensure continuity of motion.

The power and the fun of GenNets is that one can evolve a motion without specifying in detail, how the motion is to be performed. For example, the fitness (or quality) measure for the "move forward" GenNet will be a simple function of the total distance covered in a given number of cycles, (modified by a measure of its total rotation and its drift away from the desired straight ahead direction). Similar fitness measures can be defined for the two rotation GenNets. (In fact only one needs to be evolved, and one can then simply switch body side connections, for reasons of symmetry).

The "allign legs" is used to position the legs ready for either pecking or mating. Since rotations and move forward are automatically switched on and off by the input from the antenna, there is no need for timeouts, as is the case for the allign legs, peck prey and mate. It is not too difficult to design fitness measures for each of these GenNets, and since they are more or less single function GenNets (see below), their evolution should not present problems, especially after the experience gained from implementing the WALKER GenNet.

Understanding the GenNet circuit shown in FIG. 4 is fairly straightforward, except perhaps for the mechanism of rotating toward or away from the object. The antenna is fixed on the body frame of LIZZY. It is assumed in this simulation that the object is always placed initially (approximately) in front of LIZZY's body (rather than behind it). Hence if the signal strength on the left antenna is larger than that on the right, and if the object is a prey, then LIZZY is to turn towards it by rotating clockwise, or away from it by rotating anticlockwise if the object is a predator. Eventually, the two signal strengths will become approximately equal, because the two antenna will be equidistant from the object. When this happens, LIZZY moves forward. Admittedly, since FIG.4 is only a plan for future research, there will probably be conceptual and logical errors in the design. But I am confident that it will be realizable. When it does work, a video will be made of its antics. It will be interesting to see just how life like it will appear to be.

Implementing a score or so of different GenNets, (assuming a generation time of the order of a minute or less for several hundred generations per GenNet) makes one conscious of the need for Genetic Programming tools. Initially this could take the form of a software package which could ask the user for the values of the parameters needed for the GenNet to be evolved (e.g. the number of neurons, the input/output neurons, the fitness measure, the number of binary places for the weights etc) and the parameters for the Genetic Algorithm (e.g. population size, mutation probability, scaling factor etc). However, in the age of VLSI, one can readily imagine VLSI chips being designed which perform the same task as the software package, but much faster. I call such a tool, a Darwin Machine. With suitable Darwin Machines, one can imagine teams of Genetic Programmers undertaking large scale projects to build much more sophisticated nervous systems with many thousands of GenNets. One can imagine GenNet companies being established which supply Genetic Programmers with catalogues containing detailed descriptions of their companys' GenNets. In time, whole GenNet subsystems could be sold as units.

The empirical "engineering" approach to brain building ought to attract the attention of the neuro biologists. As GenNets become more biologically realistic, the solutions

found by Genetic Programmers to concrete problems of nervous system design may inspire the neuro biologists to perform new experiments. We may see a much closer collaboration between theoretical and experimental neuro biology in the near future.

Despite the attraction of the GenNet with its incredible ability to evolve almost any desired time (in)dependent output, there are some inherent limitations in the Genetic Programming approach. These limitations are of two types, intra GenNet and inter GenNet. Intra GenNet limitations concern what an individual GenNet cannot do. There is now a crying need for a new branch of theoretical (mathematical) neuro dynamics to consider what behaviours GenNets can and cannot evolve. At the moment, I am working blind, and am guided more by an empirical "Fingerspitzengefuhl" than any theoretical guidelines. Genetic Programming at the present time is very much an art.

After having played with GenNets for nearly a year now, what have I learned about intra GenNet limitations, or at least GenNet evolutionary tendencies? These lessons can be summarized as follows :-

a) Use shaping. Don't try to evolve the whole behaviour in one step - break it up (e.g. if you try to evolve a full sinusoid cycle in one go, you will probably get a straight line as a result. Instead, evolve a half sinusoid, and take the resulting GenNet as a starter for the full sinusoid).

b) When shaping, make sure your output curve has non zero slope during the last few cycles. GenNets tend to find stable (i.e. constant) outputs fairly easily, which is fine if that is what you want, but can block further evolution, if the output needs to change during later cycles.

c) Avoid multi-function GenNets. I was curious to see if it was possible to evolve a GenNet which performs two functions, e.g. to get the stick legs to move either left or right depending upon a change of input signal on one input neuron. Yes, it is possible. I got it to move both ways. But when I tried to get two separate full cycle sinusoids, I failed. I got two half sinusoids (of different frequencies) without any problem, but I could not get two adequate full sinusoids. I lost two weeks work trying to do that, and eventually gave up.

d) Keep the number of evolutionary cycles small. It is harder to evolve a desired output over a large number of cycles than over a small number. A large number increases the generation time too, of course.

e) Weight your fitness measure. If you want to "push" your evolving output curve in a given direction, weight the errors of the later cycles more heavily.

f) Chaos seems to be no problem. In the early generations, I often got what looked like chaotic output (not surprising, considering that GenNets are highly non linear, time dependent systems), but since these solutions had very low fitness scores, they were quickly eliminated.

Inter GenNet limitations are even more of a challenge. Future technologies such as (Wafer Scale Integration (WSI) [RUDNICK et al 1989], Molecular Electronics [REED 1988], Nano-technology [DREXLER 1986], [LANGTON 1989], [SCHNEIKER 1989]) and even quantum computing, will give humanity the means to build machines with [m, b, tr, ...]illions of GenNets within a human generation. How on earth are we going to know how to put them all together? Once I have finished getting LIZZY to work, I would like to try three things. The first is to give GenNets the capacity to learn. LIZZY cannot adapt its behaviour. Its reactions are merely "instinctual". The second thing is to attempt to Genetically Program robots. There is a limit to what one can do with simulations and it seems to me that Genetically Programming robots is a goal which should be aimed at. The third thing is to attempt to get the Genetic Algorithm itself, rather than a human Genetic Programmer, to connect the GenNets. At this point, the Darwin Machine could really come into its own. One can imagine populations of simulated "creatures" competing in an environment consisting of other creatures, all of which are "constructed" with the GA. The fitness measure would be truly Darwinian. Real time Darwin machines could be put into robots and a sequence of experiments undertaken on the same machine. The GA population would be derived from the stored results of each experiment in the sequence. With nanotechnology, zillions of nanorobots ("nanots"), could function in parallel in a molecular environment, and report back to some central molecular processor, which determines the next generation. Nanotechnology will probably be a reality before the end of my research career, and considering the staggering potential complexity of nanobased devices, I believe that an evolutionary approach to its foundation will be inevitable. The complexity will be such, that a system will have to be treated as a black box, whose performance only is of concern. This approach lies at the heart of Genetic Programming. I neither know nor care about what takes place inside a GenNet. I merely mutate and select, mutate and select. In a future of overpoweringly complex systems, there may be no other way. It is the approach that nature uses. Maybe we should too.

Acknowledgments

This research is supported by the Belgian National Incentive Program for Fundamental Research in Artificial Intelligence, under contract RFO/AI/19.

References

[de GARIS 1990] "Genetic Programming : Modular Neural Evolution for Darwin Machines", H. de Garis, Proceedings IJCNN90 WASH DC, International Joint Conference on Neural Networks, January 1990, Washington DC.

[DREXLER 1986] "Engines of Creation : The Coming Era of Nanotechnology", K.E. Drexler, Doubleday, 1986.

[GOLDBERG 1989] "Genetic Algorithms in Search, Optimization, and Machine Learning", D.E. Goldberg, Addison-Wesley, 1989.

[LANGTON 1989] "Artificial Life", C.G. Langton ed., Addison Wesley, 1989.
[REED 1988] "Quantum Semiconductor Devices", M.A. Reed, in "Molecular Electronic Devices", F.L. Carter, R.E. Siatkowski, H. Wohltjen eds. North Holland, 1988.
[RUDNICK et al 1989] " An Interconnect Structure for Wafer Scale Neurocomputers", M. Rudnick and D. Hammerstrom, in Proceedings of the 1988 Connectionist Models Summer School 1988, eds D. Touretzky, G. Hinton, T. Sejnowski, Morgan Kaufmann, 1989.
[RUMELHART et al 1986] "Parallel Distributed Processing", Rumelhart D.E., McClelland J.L., Vols 1 & 2, MIT Press, 1986.
[SCHNEIKER 1989] "Nano technology with Feynman Machines : Scanning Tunneling Engineering and Artificial Life", C. Schneiker, in [LANGTON 1989].

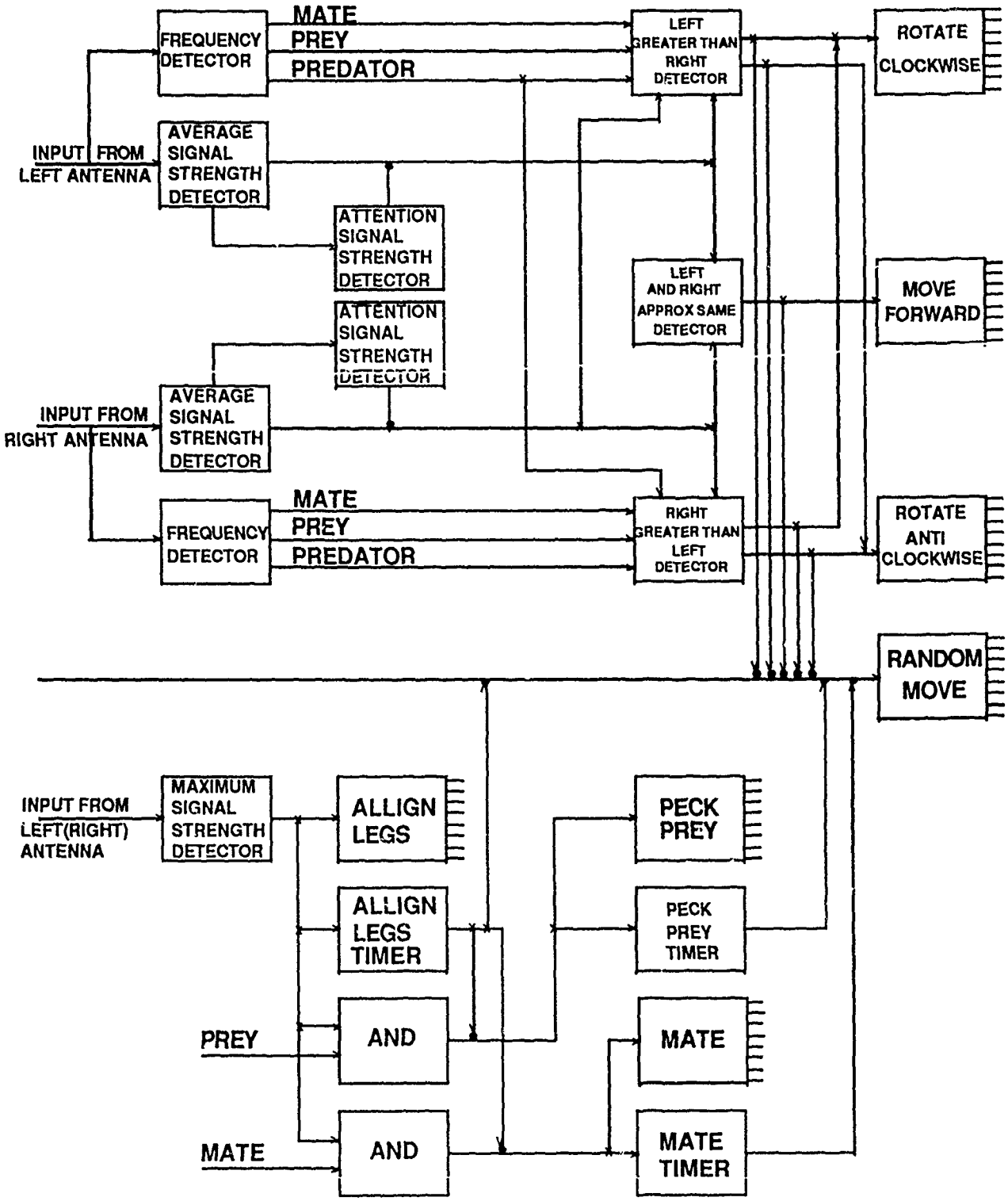

FREQUENCY DETECTOR
MATE
PREY
PREDATOR
LEFT GREATER THAN RIGHT DETECTOR
ROTATE CLOCKWISE
INPUT FROM LEFT ANTENNA
AVERAGE SIGNAL STRENGTH DETECTOR
ATTENTION SIGNAL STRENGTH DETECTOR
LEFT AND RIGHT APPROX SAME DETECTOR
MOVE FORWARD
ATTENTION SIGNAL STRENGTH DETECTOR
INPUT FROM RIGHT ANTENNA
AVERAGE SIGNAL STRENGTH DETECTOR
FREQUENCY DETECTOR
MATE
PREY
PREDATOR
RIGHT GREATER THAN LEFT DETECTOR
ROTATE ANTI CLOCKWISE
RANDOM MOVE
INPUT FROM LEFT(RIGHT) ANTENNA
MAXIMUM SIGNAL STRENGTH DETECTOR
ALLIGN LEGS
PECK PREY
ALLIGN LEGS TIMER
PECK PREY TIMER
PREY
AND
MATE
MATE
AND
MATE TIMER

FIG. 4

Improving the Performance of Genetic Algorithms in Automated Discovery of Parameters

Nagesh Kadaba

Kendall E. Nygard

Department of Computer Science and Operations Research
North Dakota State University
Fargo, North Dakota 58102-5075

Abstract

Genetic Algorithms are generally compute intensive procedures that require the evaluation of many candidate solutions to a given problem. In the application area we study (routing and scheduling), the genetic algorithm sets parameters for a mathematical heuristic. To reduce the computational overhead of this approach, we developed three mechanisms for improving the performance of the genetic search. First, we employ a method of using multiple sharing evaluation functions, permitting the parallel investigation of multiple peaks in the search space. Second, we carry out function evaluation in parallel, using a network of heterogeneous processors. Third, a neural network system is employed to inject heuristic knowledge into the initial population of the genetic algorithm, resulting in relatively fast convergence. When the methods are used together, the result is high quality solutions with considerable speedup in computational time. Our overall system, called XVTP-PGA, is a distributed software system that demonstrates the increased efficiency of genetic algorithms in the automated discovery of parameters for mathematical heuristics, in the domain of computer-aided vehicle routing and scheduling problems.

1 Introduction

In the past few years many researchers have investigated ways of improving the performance of Genetic Algorithms (GA) (Holland, 1975). GA's have shown to be useful in many optimization problems. GA's are generally compute intensive. Methods of improving the performance and efficiency of GA's are of significant importance.

The primary parameters of a standard GA are population size, crossover and mutation rates, and number of crossover points. These parameters have a significant impact on performance (Schaffer, 1989; Goldberg, 1989; Jog, 1989). Adaptive selection methods (Baker, 1985) and reproductive evaluation techniques (Whitley, 1987) have also been shown to speed up GA searches. Parallel implementations of a GA have demonstrated considerable speedup (Grefenstette, 81).

The Vehicle Routing Problem (VRP) is a highly combinatorial problem (NP-Hard) that has been extensively studied. In the VRP, there is a known collection of stop points that have demands for service, and a fixed fleet of limited capacity vehicles to serve the stops. The problem is to find the minimum distance way to assign the stops to vehicles and specify the orders in which each of the vehicles visits its stops. All the vehicles begin and end their tours at a fixed location depot. Researchers have developed many heuristic strategies for these problems (Bodin et al., 1983). A major shortcoming of these heuristics is the inability to deduce the kind of strategy to adopt, given the characteristics of the problem at hand. The need to discover the parameters to use in a heuristic for a given problem instance, led to the identification and use of GA techniques.

This paper describes ways to use GA's more effectively to discover parameters for mathematical heuristics, in the domain of computer-aided vehicle routing and scheduling problems. A method of using multiple sharing evaluation functions is described, permitting the parallel investigation of multiple peaks in the search space. Limitations of single processor systems and the increasing availability of networked heterogeneous workstations, led to investigations of utilizing the abundance of unused CPU cycles in the network in order to overcome the long search time of the GA's. In most GA's the initial population consists of entirely random structures. In routing problems, it is possible to formulate reasonably good structures for the initial population using a trained neural network system.

2 Functional Description of the Multiple Sharing Evaluation Functions

In the parameter discovery task we seek a set of parameters, under the guidance of a evaluation function. GA's use bit strings as chromosonal encodings of parameters of the problem they are trying to solve. The control parameters in our case are seed-point locations. We model each vehicle tour with a location called a seed, with the vehicle conceptually traveling from the depot to the seed and back. The seed-points represent an average location around which the truck serves. We use binary bit strings to encode the seed-points. The number of seed points recommended by the GA is equal to the number of vehicles available. The example string shown below represents one seed-point, each with an x and y coordinate, shown in both binary and decimal.

String:	0000101100	1010110010
Parameter:	S_{x1}	S_{y1}
Decoded Value:	55	803

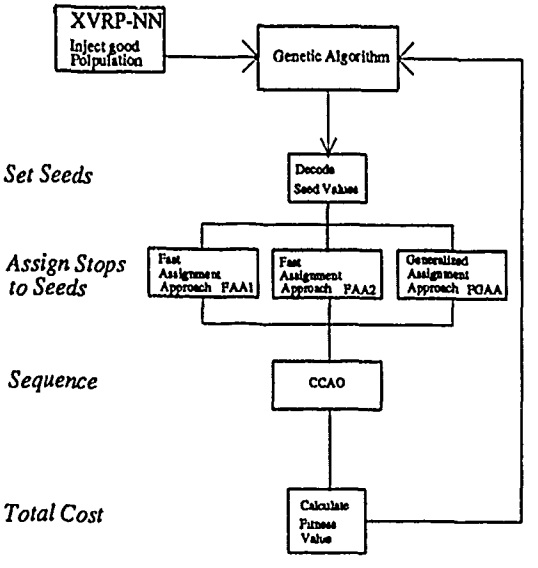


Figure 1: Unified Architecture of the overall system. The same control parameters are supplied to the three different clustering heuristics (FAA1, FAA2 and FGAA).

2.1 The Evaluation functions

We use a "cluster first route second" heuristic technique as a VRP solver in the evaluation function. There are two fundamental decisions that have to made before using this heuristic. First, we need to determine the clusters or groups of stops served by the same vehicle. Second, we need to efficiently sequence the stop locations of each vehicle. We experimented with four methods of determining the clusters. The first two methods, Fast Assignment Approaches FAA1 and FAA2, are new algorithms that are relatively fast and well suited for the genetic search. The third method, FGAA, is a modified version of the generalized assignment method developed by Fisher and Jaikumar (1981). The fourth method is a combination method, which uses the same GA recommended seed points for the three methods mentioned above. Each of the method is described below.

Clustering Method 1 [FAA1]: In this method, one seed-point is active at a time. The nearest stop is assigned to the active seed-point, if doing so does not violate the corresponding capacity constraint. For each stop assigned, a weighted distance factor is added to the active seed-point. The seed-point with the minimum weighted distance is made active for the next assignment. This process continues until all the stop points are assigned to some seed-point.

Clustering Method 2 [FAA2]: The second method FAA2 uses a simple heuristic to do the clustering. Here all the seed points are actively in the contest for receiving the next stop point assignment. The stop point with the minimum distance to any of the seed point is selected and assigned to that seed point.

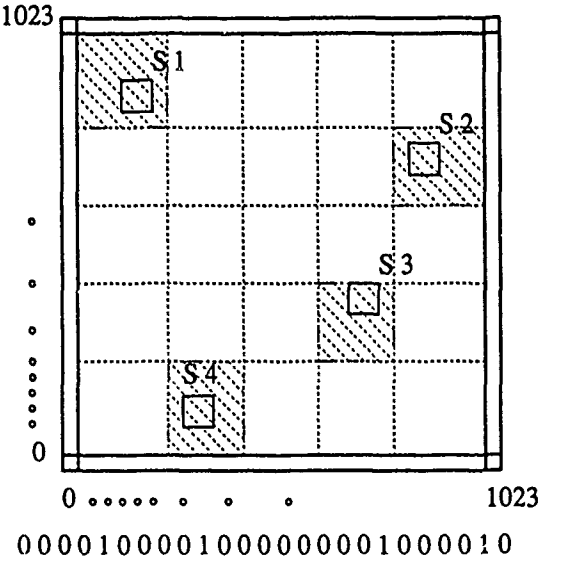


Figure 2: The 1023X1023 grid is partitioned into 25 sectors for encoding the neural network system. The seed-points are shown in the shaded regions.

Clustering Method 3 [FGAA]: A generalized assignment problem is solved to assign the stops to the seed-points (Fisher and Jaikumar, 1981).

Clustering Method 4 [COMBO method]: Many optimization problems require the investigation of multiple local optimas. Here the concept of sharing functions (Goldberg 1987) is used to investigate the formation of stable subpopulations of different strings in the GA, thereby permitting the parallel search of many peaks. This method uses the string recommended by the GA on all

three methods (FAA1,FAA2,FGAA) and the function with the best performance value is selected to return the fitness value for that particular string to the GA as shown in the Figure 1. The three methods (FAA1,FAA2,FGAA) all use the same string, and due to the competition between widely disparate points in the search space, help maintain a diverse population which searches many peaks in parallel.

2.2 The Mechanics of the Adaptive Search

The complex process of multiple vehicle routing optimization is working in an environment E. There is a set of control parameters C available for the adaptive search strategy. Within each environment there is a fitness measure for the performance of the VRP process under the control parameters. Each environment e in E to which the controlled VRP process is subjected defines a performance response surface over the control parameter space C, defined by a fitness function F_e. It is the response surface defined by F_e that is explored by the adaptive search strategy in order to generate a good performance of VRP process. In our problems, the function F_e is extremely complex, high-dimensional, multimodal and discontinuous.

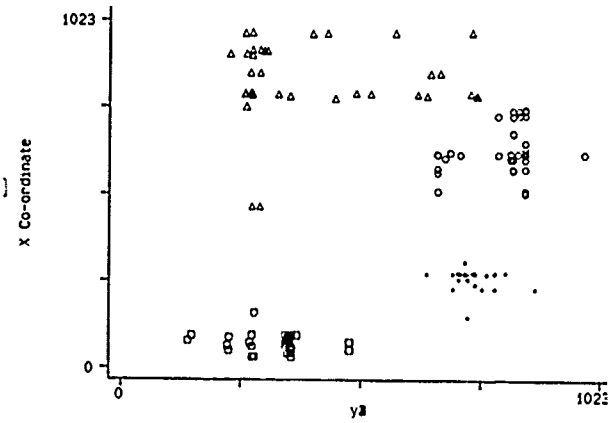


Figure 3: The Figure illustrates the activity in the last 50 trials concentrated around few good points in the search space. Each symbol defines a seed-point.

The GA exploits the accumulating knowledge of the VRP process being controlled. Each point in the control parameter space is represented as a genetic string. This string is represented in binary. In XVRP-GA, the control parameter is the location of the seed-point in 2 dimensional space. Each string has a field allocated for the performance function F_e, which is returned by the evaluation function. The GA maintains a population of these control parameter strings. Each individual is submitted for evaluation as a control parameter for the VRP process, saving the associated performance measure. Finally, using selection probabilities, these control parameter strings undergo reproduction with crossover and mutation genetic operators.

The population is collection of candidate control parameters C. Fixing one of the control parameters and leaving the other parameters free defines a hyperplane. Each parameter has (1023 X 1023) possible locations. In our case, there is one control parameter for each available vehicle. Over all possible hyperplanes, the search space is complex and multimodal. We observe the GA rapidly exploits accumulating information about F_e to restrict sampling to those hyperplanes which have a high expectation of good performance.

The search space defined by F_e is multimodal. Due to the competition between widely disparate points in the search space, using multiple sharing evaluation function helps maintain a diverse population, which could prevent a premature convergence to a local optima. FAA1, FAA2 and FGAA are all controlled by the same set of control parameters C, available form the adaptive GA. As the adaptive search progress, each of the methods is likely to be sampling different hyperplanes looking for peaks in parallel, and occasionally use the control parameters discovered by the other functions to start searching a hyperplane that is providing good performance.

Table 1, illustrates the mechanics of a crossover genetic operator. Figure 2 shows the seed-points produced by the COMBO method. The seed-point string shown in Table 1 is the value that the COMBO method uses. The encoded version of the string is used by the GA. To illustrate the effect of crossover, lets assume we are using the standard 2-point crossover method. C11 and C12 are the crossover points in Parent 1 and C21 and C22 are the crossover points in Parent 2. Now, if the genetic material between C11 and C12, C21 and C22 are exchanged, two offspring strings are generated. The use of the crossover has resulted in two candidate seed-point locations, with seed-point 3 not being effected by the crossover operation, but seed-point 1 and 2 are moved to a new location. These new location of the seed-points results in different clusters, which inturn results in a different fitness measure (total ditance). We then sequence the stop locations in each of the candidate clusters in a cost effective way (TSP) and return the result to the GA.

The first few generations start with seed-point location values uniformly distributed over the search space. As generations evolve, seed-points tend to be concentrated in tight geographical areas due to the survival of the fittest mechanism of the GA. This is illustrated in Figure 3 where only the last 50 trials are plotted. A trial is a single execution of the evaluation function on a candidate control parameter and a generation consists of many trials. The three performance curves on the graph shown in Figure 4 illustrates the survival of the fittest nature of the GA search. The performance measure is total distance traveled by the fleet, so small values are desirable. The

Table 1 The Table illustrates the mechanics of a crossover genetic operator. The Seed string shown in Table is the actual value that the COMBO method uses. The encoded version of the seed string is used by the GA in COMBO method. C11 and C12 are the crossover points in Parent 1 and C21 and C22 are the crossover points in Parent 2.

The Mechanics of the Crossover Operator						
String	S_{x1}	S_{y1}	S_{x2}	S_{y2}	S_{x3}	S_{y3}
Seed-Points	100	100	900	100	900	900
Genetic Encoding	0001100100	0001100100	1110000100	0001100100	1110000100	1110000100
Parent 1	0001100100	0001100100	c_{11}**1110000100**	**00011** c_{12} 00100	1110000100	1110000100
Parent 2	0001100100	c_{21} **0001100100**	**11100** c_{22} 00100	0001100100	1110000100	1110000100
Offspring 1	0001100100	0001100100	0001100100	1110000100	1110000100	1110000100
Decode 1	100	100	100	900	900	900
Offspring 2	0001100100	1110000100	0001100100	0001100100	1110000100	1110000100
Decode 2	100	900	100	100	900	900

curve indicates the worst performance of the evaluation function as function of generations. The bottom curve in Figure 4 indicates the best performance in each generation. The middle curve is the plot of the average performance of the evaluation function. The decreasing trend in the curves illustrates the survival of the fittest candidates in the population, and indicates that the GA is doing much better than a random walk in the control parameter search space.

Table 2 presents empirical work that illustrates the parallel nature of the search on a four vehicle problem. The values shown in Table 2 are generated when each evaluation function (FAA1, FAA2, FGAA) finds a seed-point parameter which produces a performance value better than the best found up to that point in time. The FAA1 method produces the first best solution (example 1). FAA2 then identifies a sequence of seven improving solutions, as shown in examples2 through 8. The seed-points that produce these solution are the result of searches centered around a few "good" spots. At generation 14, the GA produces a seed-point location that FAA1 adopts and improves. The coordinate value revel that this seed-point location is essentially the one FAA2 was using to improve the solution performance, as shown in example 9. The FGAA method which had not generated a best solution in earlier generations, produces one at generation 17 using seed-points from generation 2. Also note that the seed-points in example 11 are in a completely different area of the search space. This illustrates the parallel search of a multimodal response surface occurring in the algorithm.

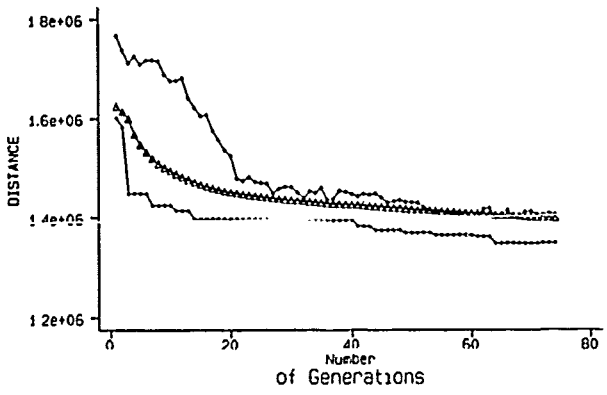


Figure 4: The three graphs illustrates the performance of the GA search in a particular generation. The middle curve indicates average performance of the evaluation.

Table 2 Parallel nature of the adaptive search. Each of the three methods is able to exploit promising seed-points locales discovered by the other methods.

Ex	S_{x1}	S_{y1}	S_{x2}	S_{y2}	S_{x3}	S_{y3}	S_{x4}	S_{y4}	Perf	Method	Generation
1	572	61	959	623	742	43	125	463	12373	FAA1	1
2	812	824	316	528	981	405	181	816	12082	FAA2	2
3	759	851	85	371	49	136	968	279	11880	FAA2	2
4	580	928	105	396	963	818	82	275	11685	FAA2	2
5	466	716	805	75	114	769	494	873	11518	FAA2	2
6	279	488	865	65	51	747	944	660	11132	FAA2	2
7	232	791	865	79	901	672	197	720	10958	FAA2	2
8	757	329	110	833	55	136	968	663	10761	FAA2	7
9	714	182	90	841	52	141	976	652	10732	FAA1	14
10	714	342	105	833	55	128	1006	648	10606	FAA2	16
11	232	151	873	185	53	795	958	464	10490	FGAA	17
12	232	150	873	185	54	868	958	464	10450	FGAA	30
13	773	181	150	185	55	868	945	431	10394	FGAA	32
14	688	197	118	185	55	868	977	524	10373	FGAA	35
15	693	169	83	838	72	151	943	908	10299	FAA2	38

3 Functional Description of the Parallel/Distributed Schema

XVRP-PGA uses the genetic algorithm's population structure to parallelize the evaluation process. XVRP-PGA is an asynchronous distributed genetic algorithm that is designed to use the idle CPU cycles on a heterogeneous network of workstations. Each workstation varies in speed, CPU architecture and operating system.

Distributed programming supports the creation of multiple processes with disjoint address spaces, and provide a means of communication among them. XVRP-PGA uses a star topology in which there is a special node called the root. Each processor is connected to the root processor which administers the network traffic. As Figure 5 indicates, this star topology does not create a bottle neck at the root processor. This is because the root processor is only doing the reproduction and selection process of the GA's and the dispatching and the polling of the results, while the remote processors are doing the compute intensive evaluation.

Remarkable speedup achieved is even with a small number of processor working in parallel because of the long computational time of the evaluation functions. At any given time only a fraction of the total CPU cycles is used in a heterogeneous network of many workstations. XVRP-PGA detects and utilizes these wasted CPU cycles and uses them in parallelizing the GA evaluation of the population. We use the Remote Procedure Call (RPC) paradigm, in which a client communicates with a server. In this process, the client first calls a procedure to send a data packets to the server. When the packet arrives, the server call the dispatch routine, performs the service requested, and sends back the reply. RPC can be used to communicate between processes on different processors as well as for communication between different processes on the same processor. RPC's are blocking, meaning the client procedure waits until the serve processor completes is task. This would defeat the purpose the parallelism. In XVRP-PGA, we use multiple daemons and immediate acknowledgment to avoid the blocking problem of the RPC.

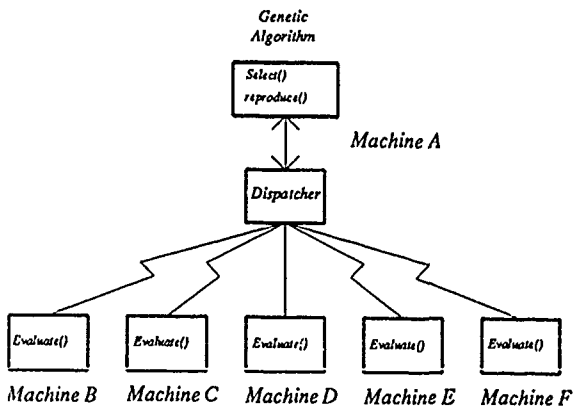


Figure 5: The heterogeneous network of workstations used for parallel evaluation of population members.

When data is being shared by two or more distinct processor types, there is need for portable data. Integers are represented differently on distinct architectures and alignment of word boundary causes the size of a data structure to vary from processor to processor. The eXternal Data Representation (XDR) standard is used to solve the data portability problem. In many parallel and distributed algorithms the time spent for interprocessor data communication is a sizable fraction of the total time required to solve the problem. The communication delays are due to the communication processing time, scheduling time, transmission time and medium (Ethernet) propagation time. However, in our experiments, the time spent evaluating each solution candidate is considerably more than the total communication delays.

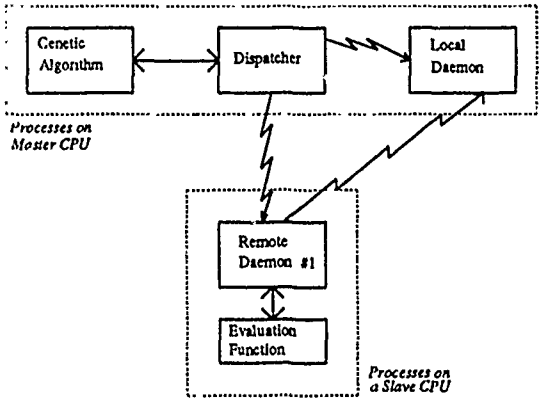


Figure 6: The communication between the master processors and the various remote slave processors, utilizing daemons.

Figure 6 illustrates the communication between the root processor and the various remote processors. The dispatcher gets a set of solution candidates selected for evaluation. This happens at every generation. The dispatcher, as shown in Figure 6, sends the decoded solution candidate to a remote site for evaluation. The remote site accepts the parameter and sends an acknowledgment. The dispatcher then sends the next candidate to the next available remote machine. After dispatching the candidate solution to all the available sites, the dispatcher polls the local_daemon for results. In the meantime, a remote site which has completed an evaluation sends the result to the local_daemon. The dispatcher receives the result from the local_daemon and immediately dispatches another candidate to the site which returned the result. In case of a remote processor malfunction, the dispatcher retransmits the candidate parameter to a functioning processor. The specific processors used in our experiments are described in Section 5.

4 Functional Description of the Neural Network Module

In this section we describe how the neural network system accumulates and preserves knowledge gained from past experiments. The neural network system described in (Kadaba et. al., 1990) is employed to inject this heuristic knowledge into the initial population. The transfer of previous experience is of significant benefit in acquiring task dependent expertise necessary to solve complex real world problems. The neural network system XVRP-NN acts as an pattern associator, matching the descriptors of the incoming problems with that of the previously solved problems. The outcome of this matching is a promising initial population for the GA. In this way, the search space is considerably pruned and the computational effort is reduced (Grefenstte, 1981; Hayong, 1985). The neural networks store parameters of previously solved problems. Figure 2 shows how the parameters are encoded. The parameters are seed-points generated by the GA. These seed-points are recorded in the course of experimentations with various problems and various environments of the GA. A potential problem in initializing a GA populations is premature convergence. To overcome this, we seed only half the population from the recommendation of the neural networks and the other half of the population with random strings. Also, a matcher first checks if the descriptors of the incoming task are similar to the training set of the neural network. The degree of similarity is the sum of squared errors between the incoming descriptors and the set of training patterns. If this measure is high, the neural network system detects no match, and the GA population is initialized only with random strings. However, due to the parallel, multi-modal nature of the GA, even highly suboptimal initial starting points in the search space injected into the population will die out in subsequent generations.

XVRP-NN basically consists of three stages each of which is described in more detail in this section The first stage consists of the Stop Data Feature Extraction (SDFE) network. In essence, a feature extraction neural network condenses the problem at hand to its essential characteristics, then passes the condensed information to a classification network that that is used for training. The teaching stimulus is identical to the input stimulus, letting the internal layers of the network adjust to "summarize" the input data. Such neural networks are also called self-organizing. All of the networks were trained with the back-propagation learning paradigm (Rumelhart and McClelland, 1986). The first stage creates and encodes the domain specific data. The problem data consists of cartesian coordinates of the stop locations. This data is pre-processed to make it invariant to scaling and rotation (Fukushima & Miyaki, 1982), producing a 30 unit problem descriptor vector. Routing problems are usually unchanged under rotations and translations of the locations

of stops to visit. However, it is well known that neural networks cannot easily discriminate among rotated and translated data sets. In general, the problem representation decision is critical.

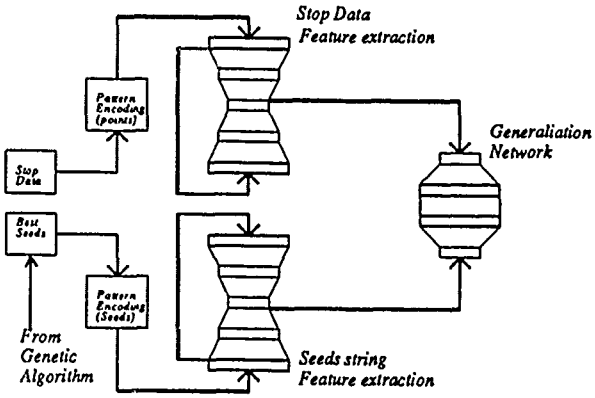


Figure 7: XVRP-NN basically consists of three modular neural networks. The Stop Data Feature Extraction (SDFE) network, the Seed String feature extraction (SSFE) network, and the generalization network GN.

The seed-point data consists of 25 values. These values represent the location of the seed-point for a given instance of a problem. As shown in Figure 7, the 25 seed-point values are normalized between 0.0 and 1.0 in the output encoding module and is used as a teaching output in the Seed-string feature extraction (SSFE) network. The second stage extracts the features from the problem descriptors and seed-point descriptors which is then fed to the generalization network GN, as training data. The SDFE vectors and the SSFE vectors are stored in two separate files. These two files are then used to train the generalizing network. Thus, there are 3 off-line trained networks (SDFE, SSFE and GN) which are subsequently used for on-line recall. The third stage is on-line generalization, resulting in the seed-point location recommendation vector.

The testing data is presented to the data encoding module, which produces a 30 unit problem descriptor vector. This problem descriptor is exposed to the SDFE network which in turn produces the 4 unit feature vector of stop data. This feature vector of stop data is then exposed to the trained generalization network. The generalizing network, on recall, produces the 4 unit feature recommendation vector, which is presented to the reconstruction network. The original dimension of 25 units in the performance vector is generalized by the reconstruction network. This vector is decoded by using a high value close to 1.0 to represent the best seed-point location for the given instance of the test stop data. Finally, the raw seed-point data is presented to the GA.

5 Experimental Results

25 problems were generated to test the system. The problems are fully dense with 200 stop points, 4 vehicles with a utilization factor of 95 %, and were generated in a square, 1023 miles on each side. The problems were run on a network of SUN, HP, VAX, and NeXT workstations. All experiments were performed using a modified GENESIS (Grefenstette, 1984) system. The Backpropagation learning paradigm (Rumelhart, 88) was employed to train the neural networks.

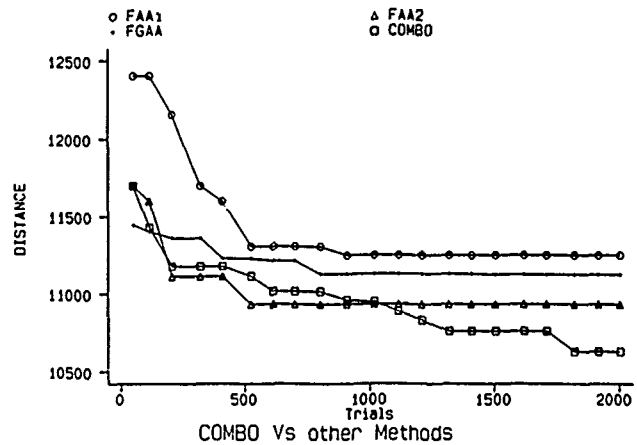


Figure 8: Comparison of performance of FAA1, FAA2, FGAA and COMBO methods for a single problem. The best solution in 1000 trials is plotted.

5.1 Performance Improvement with Multiple Sharing Evaluation Functions

In order to show the performance improvements of the COMBO method, each of the 25 problems were run for 2000 trials using the FAA1, FAA2 and FGAA methods. All the GA parameters were kept constant through out the experiments. The performance improvement of the GA by using multiple sharing evaluation functions (COMBO method) for a single problem is illustrated in Figure 8. Note, the three individual methods do not improve the search after about 1000 trials, but the performance curve of the COMBO method continues to drop through subsequent trials. This result is consistent over all 25 problems tested, and is indicated in Figure 9.

5.2 Performance Improvement with the Parallel/Distributed Schema

It is very difficult to report performance improvements of parallel algorithms when the processors are of varied strength. In a typical heterogeneous network of computers, the load on each processor at any give time is very different. The dispatcher we designed does not preempt other processes in the network but utilizes the unused CPU cycles of the computers on the network. The network consisted of various processors, including SUN 3/260, SUN 3/110, SUN 3/50, VAX 11/780, NeXT, HP 9000, and AT&T 3B2. Figure 10 shows the speed up achieved in running 1000 trials using Method_1, when we start with the weakest processor and add the stronger processors. Method_2 starts with a strong processor. Note that for a given population size, there is a limit on the number of remote processors we can use. The parallel algorithm is robust and considerable speed up is achieved without affecting the GA search mechanism.

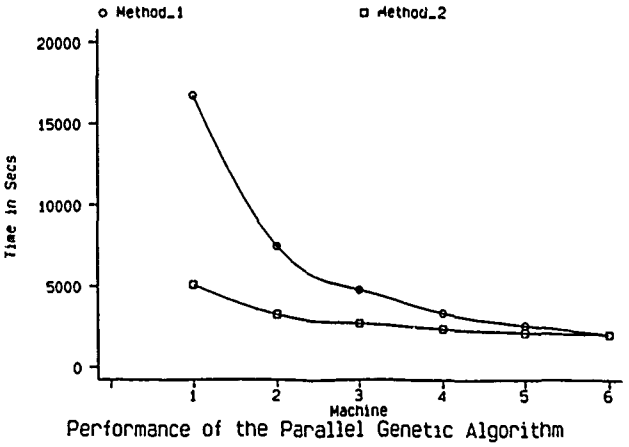


Figure 10: Performance improvement with the parallel implementation. The value plotted is the CPU time required to run 1000 trails with the COMBO method. In Method_1 the weaker processors are added and in Method_2 the stronger processors are added.

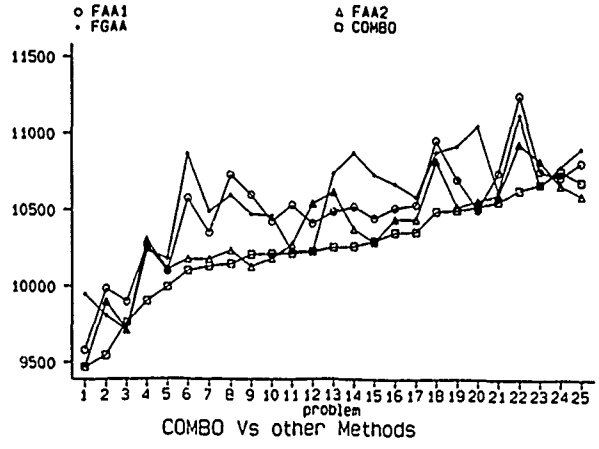


Figure 9: Multiple evaluation functions results. The COMBO method consistently performs better than that achieved using any single evaluation function.

5.3 Performance Improvement with the Neural Network Initialization

The neural network system is employed to inject heuristic knowledge into the initial population. Figure 11, shows the improvement that GA search received on a typical problem when its initial population was seeded with the recommendations from the neural network system. The seeded method finds a high performance parameter initially, and continues to improve during subsequent trials.

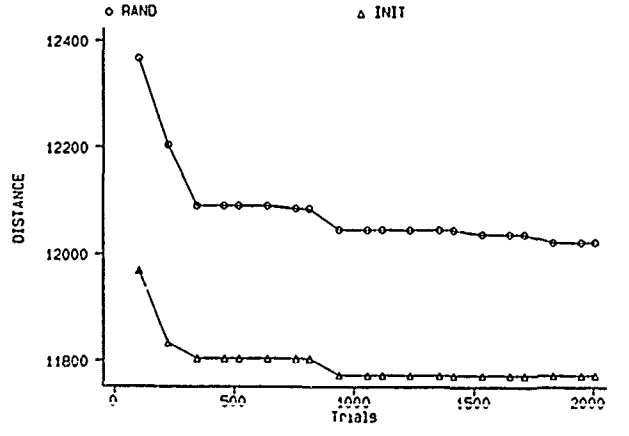


Figure 11: Performance improvement achieved on a single problem when the initial population of the genetic search is seeded with the recommendations from the neural network system.

6 Conclusion

The results demonstrate considerable impro . in the GA search by using multiple sharing evaluation functions. Seeding the initial population with heuristically chosen population structures using a neural network system also achieves a significant speedup. The heterogeneous network of computers is a growing trend in many research institutions. It is often the case that only a fraction of the total CPU cycles is used in such a network. XVRP-PGA utilizes CPU cycles throughout the network and uses them in parallelizing the evaluation of the population structure within the genetic search. Each of the methods described in this paper can be used separately or together. When all three are used, relatively powerful genetic search can be conducted for parameter discovery, in an environment of networked desktop workstations.

7 References

Baker, J.E., *Adaptive Selection Methods for Genetic Algorithms,* **Proceedings of the First International Conf. on Genetic Algorithms and their Applications,** July 24-26, Pittsburgh, 101-111, 1985.

Bodin, L., Golden, B.R., Assad, A., and Ball, M., *Routing and Scheduling of Vehicles and Crews,* **Computers and Operations Research** 10, 62-212, 1983.

Fisher, M., and Jaikumar, J., *A Generalized Assignment Heuristic for Vehicle Routing,* **Networks** 11, 109-124, 1981.

Goldberg, D., *Genetic Algorithms with Sharing for Multimodal Function Optimization,* **Proceedings of the Second International Conf. on Genetic Algorithms ,** July 28-31, MIT, 41-49, 1987.

Goldberg, D., **Genetic Algorithms in Search, Optimization and Machine Learning,** Addison-Wesley, 1989

Goldberg, D., *Sizing Population for Serial and Parallel Genetic Algorithms,* **Proceedings of the Third International Conf. on Genetic Algorithms ,** June 4-7, George Mason University, 70-79, 1989.

Grefenstette, J., *Parallel Adaptive Algorithms for Function Optimization (Preliminary Report),* Technical Report CS-81-19, Computer Science Department, Vanderbilt University, 1981.

Grefenstette, J., *GENESIS: A system for using genetic search procedures,* **Proceedings of the 1984 Conf. on Intelligent Systems and Machines,** 161-165, 1984.

Hayong, Z., *Classifier System with Long-term Memory in Machine Learning* **Proceedings of the First International Conf. on Genetic Algorithms and their Applications,** July 24-26, Pittsburgh, 178-182, 1985.

Holland, J., **Adaptation in Natural and Artificial Systems,** Ann Arbor, MI: University of Michigan Press, 1975.

Jog, P., Suh, J.Y., and Gucht, D.V. *The Effects of Population Size, Heuristic Crossover and Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem,* **Proceedings of the Third International Conf. on Genetic Algorithms ,** June 4-7, George Mason University, 110-115, 1989.

Kadaba, N., Nygard, K.E., and Juell, P.L. *Integration Of Knowledge-Based System And Adaptive Learning Techniques For Routing And Scheduling Applications,* forthcoming in **Expert Systems with Applications: An International Journal,** 1990.

Kadaba, N., Nygard, K.E., Juell, P.L. and Kangas, L., *Modular Back-Propagation Neural Networks for Large Domain Pattern Classification,* **Proceedings of the International Joint Conf. on Neural Networks,** Washington D. C., January, II-551 - II-554, 1990.

Kadaba, N., Perrizo, W., Ram, P., *Performance Analysis of Distributed Systems for Parallelizing Genetic Algorithms,* **NDSU Technical Report NDSU-CS-TR-90-4, 1990.** Submitted for publication.

Kadaba, N., Nygard, K.E. *An Effective Heuristic using Genetic Algorithms for Computer-Aided Vehicle Routing.* **NDSU Technical Report NDSU-CS-TR-90-11,** Submitted for publication.

Kadaba, N., *XROUTE: A Knowledge-Based Routing System using Neural Networks and Genetic Algorithms,* **Ph.D. Dissertation,** Department of Computer Science and Operations Research, North Dakota State University, Fargo, 1990.

Rumelhart, D. E., and McClelland, J., **Parallel Distributed Processing, Explorations in the Microstructure of Cognition,** MIT Press, Cambridge, 1986.

Schaffer, D., Caruana, R.A., Eshelman, L.J., and Das, R. *A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization,* **Proceedings of the Third International Conf. on Genetic Algorithms ,** June 4-7, George Mason University, 51-60, 1989.

Whitley, D., *Using Reproductive Evaluation to Improve Genetic Search and Heuristic Discovery,* **Proceedings of the Second International Conf. on Genetic Algorithms ,** July 28-31, MIT, 108-115, 1987.

Using Genetic Algorithms to Learn Disjunctive Rules from Examples

R. Andrew McCallum
Biomedical Information Communication Center
Oregon Health Sciences University
Portland, Oregon
andrewm@ohsu.edu

Kent A. Spackman
Biomedical Information Communication Center
Oregon Health Sciences University
Portland, Oregon
spackman@ohsu.edu

Abstract

In this paper we address the problem of learning disjunctive normal form rules from boolean-classified examples. Whereas GA's often present their solution in the form of a single individual, here we use the entire population of individuals to disjunctively represent a solution. We explain *example sharing*, a method of payoff sharing, and we explain *over-population*, a modification for increasing GA exploration.

1 Introduction

Problems with disjunctive solutions do not lend themselves to the traditional Genetic Algorithm (GA). Typically, the successful Simple Genetic Algorithm produces as its result the one individual representing the best evaluated point in the parameter space. Since individuals usually have constant length and disjunctions are by nature variable in length, some departure from the Simple GA is necessary. Disjunction might be achieved by either "messy" GA's (using variable-length individuals) [Goldberg *et al.*, 1989], or by methods that use multiple individuals.

The last approach was taken by Wilson's classifier system, called "Boole", [Wilson, 1987]. The "Boole" classifier system learned disjunctive boolean concepts by addressing the animat problem—thus, "Boole" operated under the constraints of learning incrementally and of noisy and/or partial feedback.

For many inductive learning tasks, however, these constraints do not hold, and we can learn non-incrementally with a database of correctly classified examples. In this case, the classifier approach seems to complicate the model unnecessarily.

We present a model based on the Simple GA which learns disjunctive concepts. The essential enhancements of our model are 1) a niche-formation method which utilizes a novel version of payoff sharing, and 2) an exploration-increasing scheme of population management.

Our GA produces as its result the entire population. We let each individual represent a conjunction of variables, and then take the disjunction of the whole population to produce a single boolean answer. In this way the GA's population represents a disjunctive normal form boolean formula.

The individuals are comprised of genes which may take on the alleles true, false and don't-care. There is a one-to-one correspondence between the genes of an individual and the boolean variables of the parameter space to be searched. If the gene string of an individual, acting as a template, "matches" the boolean value string of an example, then that individual includes or *covers* that example. Thus, each individual represents the conjunction of its *non*-don't-care variable values.

The question becomes, how does one get a GA to learn a population of templates that will disjunctively cover the positive examples to be learned, but avoid covering the negative examples. One obvious solution is to use a niche-formation approach to prevent convergence of the population to a single individual. We accomplish this niche-formation by *example sharing*.

2 Example Sharing

Payoff sharing is a well-known method of niche-formation. Goldberg and Richardson implemented sharing by decreasing an individual's payoff according to its proximity to other individuals [Goldberg and Richardson, 1987]. This method successfully produced clusters of individuals on the peaks of the evaluation function, thus performing niche-formation [Deb and Goldberg, 1989].

Niche-formation behavior is what we need. We want individuals to cover clusters of positive examples (the peaks) while avoiding covering the negative examples (the valleys). In this way clusters of positive examples in the parameter space are desirable niches.

2.1 Definition of Example Sharing

Instead of sharing based on a proximity measure, however, we find that this problem lends itself to a more directly applied form of sharing — individuals share the value of each example they cover with other individuals that also cover the same example. Instead of using a measure based on the geometry of the parameter space, independent of the specific examples to be learned, this enforces direct dependence on the distribution of these examples. Conceptually this sharing

performs equally well in areas of the parameter space where clusters of positive and negative examples are small and close together as in areas where the example clusters are expansive and widely spaced. It is not based on a constant distance factor, but varies with the layout of the positive and negative examples. This method is much like Wilson's distribution of environmental feedback among the action set of classifiers.

We begin the calculation of example sharing payoff as follows: The positive examples each have a positive value, and the negative examples each a negative value, i.e. 1 and -1. Determine the number of individuals that cover each example, and divide the example's value by this number. The result is the payoff from this example (e) that is given to each individual (i) covering it. One can intuitively think of each example as having only a finite amount of payoff to give, and this finite payoff being *shared* equally among its recipients.

$$\text{payoff}_{ie} = \begin{cases} 0, & \text{if } i \text{ doesn't cover } e \\ \frac{\text{value}_e}{\#\ i\text{'s covering } e}, & \text{if } i \text{ covers } e \end{cases}$$

Then the fitness of an individual is

$$\text{fitness}_i = \sum_{\text{all } e} \text{payoff}_{ie}$$

2.2 Adjusted Payoff

There is, of course, one immediate problem with the above calculation. It is possible for an individual which covers negative examples to receive a negative payoff value, confounding completely the "roulette wheel" method of selection. The probability that a certain individual will be chosen at a single spin of the wheel is

$$\text{p}(i \text{ reproduce}) = \frac{\text{fitness}_i}{\sum_{\text{all } i} \text{fitness}_i}$$

Clearly, an individual with a negative fitness produces a negative probability—an impossibility. In order to properly decide the probability of an individual's selection according to its evaluation, all evaluation values must be positive.

Giving negative examples a value of zero is not a viable solution, since this would give no penalty to individuals which cover them. Instead, we pass the above calculated fitness through a function which makes negative numbers positive while roughly keeping the proper relation between the individuals' fitness and their probabilities of selection.

$$\text{adjusted fitness}_i = \begin{cases} \text{fitness}_i + 1, & \text{if fitness}_i \geq 0 \\ e^{\text{fitness}_i}, & \text{if fitness}_i < 0 \end{cases}$$

2.3 Specificity Factor

There is still a problem with our approach thus far. The above calculations count equally both positive and negative examples. How many positive examples do we want an individual to gain in order to justify covering an additional negative example? The implementation of the above model could conceivably rank equally an individual that covered three positive examples and no negative examples with an individual that covered four positive examples and one negative example. Is this what we want? Well, that depends on how much specificity versus sensitivity and simplicity we want. If we penalize an individual heavily for covering a negative example, the GA will tend to produce highly specific individuals. This may be desirable, but what we gain in specificity we may lose in sensitivity (the population will tend not to be able to cover all positive examples), and lose in simplicity (the population will be more diverse, have less duplication of individuals, and therefore translate into longer disjunctive normal form formulas). Thus, we introduce a user-settable *specificity factor* by which we multiply the values of the negative examples. A specificity factor of one will make no change from the above calculations; increasing the factor will tend to create more and more specific individuals.

3 Increasing GA Exploration

In addition to modifying the sharing mechanism used to develop niche-formation, we have made some changes to improve the exploration of the GA so that fewer positive niches will be left uncovered. Increasing the probability of mutation and the probability of crossover have the obvious effect of increasing the exploration of a population, but unfortunately they also tend to introduce lethals and remove desirably fit individuals. We have developed a modification we call *over-population* which increases exploration, yet alleviates this problem.

3.1 Over-population

Over-population is a population management strategy somewhat like population overlap [DeJong, 1975]. In population overlap a certain percentage of the individuals in a population are chosen to skip crossover and mutation, and pass unchanged to the next generation. In over-population the *entire* population is passed on unchanged. How does any innovation take place, you ask? Each individual of the population *does* go through crossover and mutation to produce new individuals for the next generation—but the parents and unmutated individuals also remain in the population. In this way, each operation doubles the population size.

For example, an initial population of 20, after mating, will contain both the parents and the children, for a total of 40 individuals. Mutation of this set of individuals will produce both the mutated parents and children, and the unmutated parents and children, for a total of 80 individuals. Payoff sharing now occurs among all 80 individuals. This quadrupled population is brought back to its original size of 20 in the selection step, which only spins the fitness roulette wheel 20 times.

This procedure can be thought of as a global best-preserving algorithm. It allows us to make all individuals mate (p(cross) = 1.0, no mating restriction), and all individuals mutate (p(gene change) = $1/\text{length}_{\text{individual}}$), in order to increase GA exploration to the maximum, while it also retains the best of both

the pre- and post-exploration population for consideration in selection. Intuitively one can imagine a situation where the birthrate far exceeds the sustaining support of the environment, and individuals are dying off not from old age, but from competition with each other—thus, we say over-population.

3.2 Over-population Performance

This method quadruples the number of individuals to be evaluated, but the increased exploration makes the GA converge to a solution more quickly. We performed experiments using the AP. 10 data set of bacterial classification [Robertson and MacLowry, 1975]. The bacteria are described by ten binary attributes and are classified as either *e coli* or not *e coli*. There are 37,555 examples of 276 different 10-bit patterns, the more common bacteria being replicated proportionally more times. Using a population size of 30 and a specificity factor of 3, we ran the example-sharing GA both *with* and *without* over-population ten times each. The median number of generations required to learn the bacterial classification to 98% accuracy *with* over-population was 5 (range 1 to 26) versus 46 (range 4 to 138) *without* over-population.

We believe that enhanced exploration resulting from over-population is especially helpful in situations such as this, where not all of the points in the parameter space have a specified value.

In addition to faster convergence, we expect over-population to allow the use of smaller populations than are otherwise necessary to keep genetic diversity. We have not yet shown this experimentally.

4 Comparison with "Boole" —Experimental Results

In this section, we present the performance of our GA on examples from a disjunctive boolean function, and compare the results with "Boole."

4.1 The Example Set

The examples to be learned conform to the "6-multiplexer" function also used by Wilson [Wilson, 1986]. The function can be described as follows. Of the six bits of the parameter space, the first two bits can be thought of as address bits, while the last four make up the data bits. The value of an example is determined by the value of the data bit found at the location specified by the address bits. For instance, the example (0 0 1 0 0 0) would be classified as a positive example since the first two bits specify address zero and the zeroth data bit is a one. The example (1 1 0 1 1 0) would be classified as a negative example because the third bit is a zero.

For instance, one possible set of disjuncts that covers all the positive examples and no negative examples of the six-multiplexer is:

```
0 0 1 * * *
0 1 * 1 * *
1 0 * * 1 *
1 1 * * * 1
```

4.2 Implementation Details

The example-sharing GA with over-population was implemented in Allegro Common Lisp on a NeXT cube. The following simulation results were obtained in an average of 60 seconds of CPU time.

4.3 Simulation Results

We set the population size to 50 and the specificity factor to 3. Note that example sharing inherently specifies a p(cross) of 1.0 and p(gene change) of 1/individual length. Uniform crossover was used for mating [Syswerda, 1989]. Based on ten runs, the example sharing GA achieved perfect performance (100% accuracy) on the 6-multiplexer example set after a median 5 generations, or 1000 trials. This compares very favorably with Wilson's 12,000 trials and 97.3% accuracy [Wilson, 1986].

5 Future Work

One question, as yet unresolved, is whether this approach to learning disjunctive concepts has any advantage over known methods such as AQ [Michalski *et al.*, 1986] and C4 (ID3) [Quinlan, 1986].

We did some preliminary comparisons with C4 using the same bacterial classification data previously mentioned in section 3.2. The example-sharing GA with over-population converged after a median of 5 generations to a simple 2-term solution that has 99% accuracy. C4, after pruning, produces a tree with 10 nodes and 11 leaves with 99.5% accuracy. Thus our GA produces a simpler representation with slightly less accuracy. Changing the specificity factor would very likely alter the accuracy and simplicity.

Efficient performance is another concern. Scale-up may be better because of the implicit parallelism of GA's. However, Quinlan's assessment of "Boole" indicated that C4 and "Boole" scaled approximately equally [Quinlan and Compton, 1987].

Furthermore, the global nature of our example sharing (considering the entire population and all the examples together) may permit this approach to find more desireable DNF rules than would be found using the more local search involved in decision-tree algorithms or AQ-like approaches.

6 Conclusions

This paper presents two modifications to the Simple GA that accomplish the learning of disjunctive boolean concepts. *Example sharing* has been shown to successfully form niches over clusters of positive examples. We have introduced *over-population* as a method of increasing exploration and have shown that it can improve the successful convergence rate by a factor of nine. Our modified GA successfully learned the 6-multiplexer boolean function in one-twelfth the number of trials as "Boole." Additional work will be needed to determine its value as a general-purpose inductive learning method.

Acknowledgments

This research is supported in part by grant no. R29 HL41581 and contract N01 LM43501 from the National Institutes of Health, Bethesda, MD.

References

[Deb and Goldberg, 1989] K. Deb and David E. Goldberg. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[DeJong, 1975] K. A. DeJong. *An Analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975. University Microfilms No. 76-9381.

[Goldberg and Richardson, 1987] David E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms.*, 1987.

[Goldberg *et al.*, 1989] David E. Goldberg, Bradley Korb, and Kalyanmoy Deb. Messy genetic algorithms: Motivation, analysis, and first results. Technical report, Department of Engineering Mechanics, University of Alabama, May 1989. TCGA Report No. 89003.

[Michalski *et al.*, 1986] Ryszard S. Michalski, Igor Mozetic, Jiarong Hong, and Nada Lavrac. The multi-purpose incremental learning system AQ15 and its testing applications to three medical domains. In *Proceedings AAAI-86*, 1986.

[Quinlan and Compton, 1987] J. R. Quinlan and P. J. Compton. Inductive knowledge acquisition: a case study. In J. R. Quinlan, editor, *Applications of Expert Systems*, chapter 9, pages 157–173. Addison Wesley, Reading, Mass., 1987.

[Quinlan, 1986] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1), 1986.

[Robertson and MacLowry, 1975] E. A. Robertson and J. D. MacLowry. Construction of an interpretive pattern directory for the API-10S kit and analysis of its diagnostic accuracy. *Journal of Clinical Microbiology*, 1:515–520, 1975.

[Syswerda, 1989] Gilbert Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[Wilson, 1986] Stewart W. Wilson. Quasi-darwinian learning in a classifier system. In *Proceedings of the Fourth Internation Workshop on Machine Learning*, 1986.

[Wilson, 1987] Stewart W. Wilson. Classifier systems and the animat problem. *Machine Learning*, 2(3), 1987.

NEWBOOLE: A Fast GBML System

Pierre Bonelli
bonelli@lri.lri.fr
Equipe Inférence et Apprentissage
LRI
bât. 490
Université de Paris-Sud
91405 Orsay, France

Alexandre Parodi
parodi@lri.lri.fr
INFODYNE SARL
10, rue de la Paix
75002 Paris
France
Tel.: (33)-(1)-42-61-56-08

Sandip Sen
sandip@dip.eecs.umich.edu
EECS Department
University of Michigan
Ann Arbor, MI 48109
USA

Stewart Wilson
wilson@think.com
The Rowland Institute for Science
100 Cambridge Parkway,
Cambridge, MA 02142
USA

Abstract[1]

Genetics based machine learning systems are considered by a majority of machine learners as slow rate learning systems. In this paper, we propose an improvement of Wilson's classifier system BOOLE that shows how Genetics based machine learning systems learning rates can be greatly improved. This modification consists in a change of the reinforcement component. We then compare the respective performances of this modified BOOLE, called NEWBOOLE, and a neural net using back propagation on a difficult boolean learning task, the multiplexer function. The results of this comparison show that NEWBOOLE obtains significantly faster learning rates.

1 Introduction

In recent years, Genetics Based Machine Learning (GBML) has received increasing attention from the ML community due to the emergence of Classifier Systems. However, despite the demonstrative results obtained by various researchers with Classifier Systems (CS) [Goldberg, 89], the slow learning rates that are usually observed have considerably affected their credibility. The results reported in this paper, using an improvement of Wilson's BOOLE system, tend to show that convergence speed of GBML systems can be greatly improved.

[1] This research was partially supported by MRT through PRC IA.

2. Slow learning rates in GBML systems.

A number of realizations in the domain of CS have shown their undisputed ability to learn. Classifier Systems [Holland, 86] form a family of inductive learning systems which acquire *rules incrementally*. However, these realizations all share the common drawback of slow rate learning when compared to other widespread learning algorithms such as decision tree classification (such as ID3) or neural net back propagation.

Wilson's classifier system BOOLE [Wilson, 87] is an example of such a realization. BOOLE is an *incremental* learning system that learns intricate boolean functions such as logic multiplexers. However, more recently, Quinlan [Quinlan, 88] compared the respective performances of an improved version of the ID3 algorithm (C4) and BOOLE on the multiplexer problem, and evidenced a much faster convergence rate with C4. We show in this paper that this drawback can be greatly weakened by modifying the reinforcement component of the original algorithm. Furthermore, C4 *is non incremental*, and as Booker mentions in [Booker, 89], having access to all the examples at once is a definite advantage. Therefore, in our experiments, we decided to compare the improved version of BOOLE (NEWBOOLE) with a widely used incremental learning system: a neural net using back-propagation.

3. The BOOLE Classifier System

We now present BOOLE with some detail concerning the parts that have been changed, in order to explain the NEWBOOLE system in the following section.

BOOLE is a simplified version of the standard Classifier System (CS) which was designed by Wilson to test the ability of a GBML system to learn difficult boolean functions.

Like any CS, Boole maintains a population of classifiers (which can be thought of as bit-level zero order rules) according to Darwinian evolution principles. However, classifiers are not chained; they directly provide an output and the decision is made within a single step during recognition; consequently there is no message list nor Bucket Brigade Algorithm. Thus each classifier consists of a condition (taxon) and an action which are fixed length strings over the {0,1,#} alphabet.

Like other CS, BOOLE has the following components:

1/ **Performance component**: in the performance cycle, an input string is presented to the system, the match set M of all classifiers whose taxa match the input string is formed, and a single classifier from M is selected (using a probability that is proportional to its strength) whose action is output as the system's decision.

2/ **Reinforcement component**: this component modifies the strengths of classifiers according to performance level:

a/ Form the action set [A] consisting of classifiers from [M] whose action is the same as the chosen action; the remaining members of [M] form the set Not[A];

b/ Deduct a fraction e from the strengths of all classifiers in [A];

c/ * If the system's decision was correct, distribute a payoff quantity R to the strengths of [A]; but
* If the decision was wrong, distribute a payoff quantity R' (where $0 \leq R' < R$) to the strengths of [A] and deduct a fraction p from the strengths of [A] (at least one of R' and p is equal to 0);

d/ Deduct a fraction t from the strengths of Not [A].

The distribution of payoff is done so that rules which have many # 's (thus more general) are favored.

3/ **Discovery component**, which modifies the classifier population according to Holland's genetic algorithm [Holland, 75] and employs reproduction, genetic operations (crossover and mutation), and deletion.

BOOLE's version of the genetic algorithm is quite particular in the sense that only one offspring is added per invocation of the genetic algorithm. In this context, the parameter ρ will represent the average number of invocations of the genetic algorithm per cycle (i.e. the number of offspring added per cycle). For the detailed algorithm, please see [Wilson, 1987].

Wilson experimented in [Wilson, 87] with this system using a highly disjunctive function, the multiplexer function, also used by Barto [Barto,1985]. In the case of the "6-multiplexer", for each six-bit input string ($a_0, a_1, x_0, x_1, x_2, x_3$), the boolean expression of the output is:

$$F_6 = \neg a_0 \cdot \neg a_1 \cdot x_0 + a_0 \cdot \neg a_1 \cdot x_1 + \\ + \neg a_0 \cdot a_1 \cdot x_2 + a_0 \cdot a_1 \cdot x_3 \qquad (1)$$

Figure 2 shows an experiment in which BOOLE learned to respond correctly to this problem. The parameters used for this experiment are given in section 4.

At each cycle, an example is chosen at random and is presented to the system. The graph plots the system's *average score* which is the percentage of correct decisions over the past 50 cycles versus the number of cycles since the experiment began.

The results obtained by BOOLE show that a "rather difficult disjunctive incremental learning task" can be solved by GBML. However, the learning rate is extremely slow, as was pointed out by Quinlan in [Quinlan, 88], where he compares the respective performances of BOOLE and C4 on the same multiplexer task.

4. The NEWBOOLE CS

NEWBOOLE is a CS derived from Boole which obtains much faster learning rates. We examine in this section the improved learning algorithm.

4.1 A new payoff strategy: "Symmetrical payoff-penalty

BOOLE's reinforcement component, under the "payoff-penalty" reinforcement regime ($p \neq 0$) adjusts classifier strengths in the following way:

- if the system's decision is correct, distribute a quantity R to the strengths of the Actionset [A].

- if the system's decision is false, penalize the strengths of [A] by deducting a fraction p from their values.

- finally, whether the system's decision is correct or not, deduct fractions e and t respectively from the strengths of [A] and Not[A].

Thus, following each performance cycle, *only the strengths of [A] are ajusted according to the system's performance.*

However, once we know that [A] contains accurate classifiers, we also know that Not[A] only contains inaccurate classifiers; in this case it would make sense to penalize the rules in Not[A]. This acknowledgement led us to a "symmetrical payoff-penalty" algorithm, in which we respectively reward and penalize the accurate and inaccurate classifiers present in the Matchset.

The new reinforcement component is the following:

1/ Form the subset of [M] consisting of those classifiers whose action is accurate; this is the correct set [C]. The remaining members of [M] form the set NOT[C].

2/ Deduct a fraction e from the strengths of [C].

3/ Since [C] contains the accurate classifiers, distribute a payoff quantity R to the strengths of [C].

4/ Since Not[C] contains the inaccurate classifiers, deduct a fraction p from the strengths of Not[C].

Thus, the effect of the reinforcement component can be written as:

$$S_{[C]}(t+1) = (1-e) \times S_{[C]}(t) + R \qquad (3)$$

$$S_{Not[C]}(t+1) = (1-p) \times S_{Not[C]}(t) \qquad (4)$$

where $S_{[C]}$ and $S_{Not[C]}$ are respectively [C]'s and Not[C]'s total strengths.

This new algorithm constitutes a clear departure from Boole: indeed, if we have several possible output values then the knowledge of the correctness of the output of each classifier from the match set is used. This information can be provided by the knowledge of the correct output for each example, as is done in most learning systems. However, this does not make any difference with boolean functions such as the Multiplexer since only two values are possible: if one is known as wrong, then the other one is right.

As in Boole, the payoff R to [C] is distributed by a biased distribution function D, which favors more general rules (i.e. with many "don't cares" #) as follows.

First, the generality of each classifier i of length L is computed as:

$$g_i = \frac{\text{number of \#'s in i}}{L} \qquad (5)$$

Let us define:

$$d_i = 1 + G \times g_i \qquad (6)$$

where G is a "generality emphasis" parameter.

Then, the portion of reward R_i that is given to classifier i becomes:

$$R_i = D(i) \times R = \frac{d_i}{\sum_i d_i} R \qquad (7)$$

4.2 Experiments with NEWBOOLE

We experimented with NEWBOOLE using the multiplexer problem, our main concerns being on the one hand to compare BOOLE's and NEWBOOLE's respective performances, and on the other, to compare NEWBOOLE and a neural net using Back Propagation (BP). We describe two sets of experiments, one with the 6-multiplexer, the other with the 11-multiplexer.

Each experiment was conducted by making 4 independent runs with different random initializations and averaging the values over these runs.

The table below gives a complete description of the genetic experimental parameters used.

4.2.1 NEWBOOLE and the 6-multiplexer problem.

1/ In the first experiment (Figures 1, 2), we tested NEWBOOLE's performance using exactly the same parameter values as in BOOLE in order to evaluate the effect of the change in the reinforcement component. As it can be seen, the results are quite demonstrative: without any "parameter tuning", we are able to enhance importantly the system's learning rate to 97.3 % after only 2000 trials. The learning rate only takes into account the system's performance.

The lower plot in Figure 1 shows a quantity called the relative solution count, equal to the relative number of instances of the *solution set* [S6], which represents the minimal set of classifiers capable of solving the problem perfectly.

Param. / Experi.	e	χ cross-over rate	μ mutation rate	G generality enforcement	R reward	p penalty coef.	ρ renewing	t	P population	deterministic output
Boole 6Mux Fig1,2	0.1	0.125	0.001	4	1000	0.8	1	0.1	400	NO
NewBoole 6Mux (Fig 1, 2)	0.1	0.125	0.001	4	1000	0.78[1]	1	////	400	NO
NewBoole 6Mux (Fig. 3)	0.1	0.5	0.001	4	1000	0.95	4	////	400	YES
NewBoole 11 Mux (Fig. 4)	0.1	0.5	0.001	4	1000	0.95	4	////	1000	YES

Table 1: Experimental parameters for Boole and NewBoole.

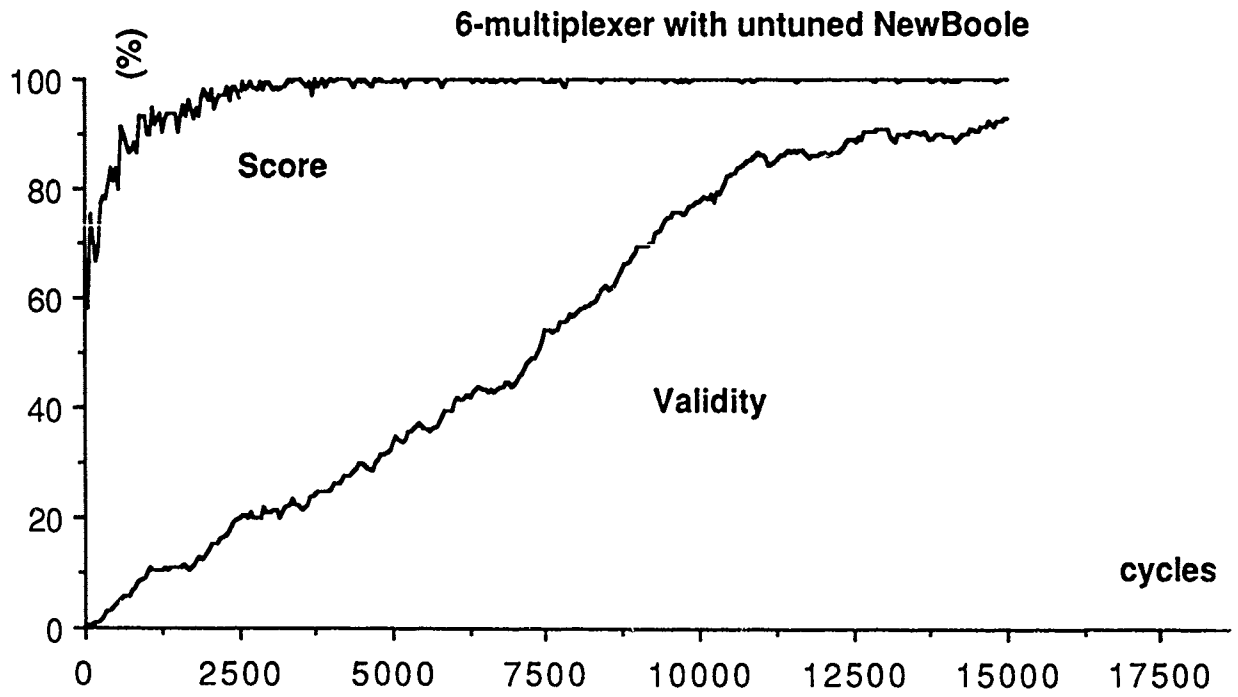


Figure 1: Untuned New Boole with stochastic output

[1] p is taken as 0.78 and not 0.80 in order to ensure total equivalence between the experiments with Boole and untuned Newboole, since the parameter t is no longer used in the Newboole algorithm.

Hence, this ratio is a measure of the *validity* of the population: the predominance of [S6] in the evolved population would show that the system is capable of finding the best among the accurate classifiers.

It is interesting to notice that eventhough the system attains quasi-perfect response (99.8) after 3200 cycles, the validity (solution count) continues to grow at a similar rate than in BOOLE.

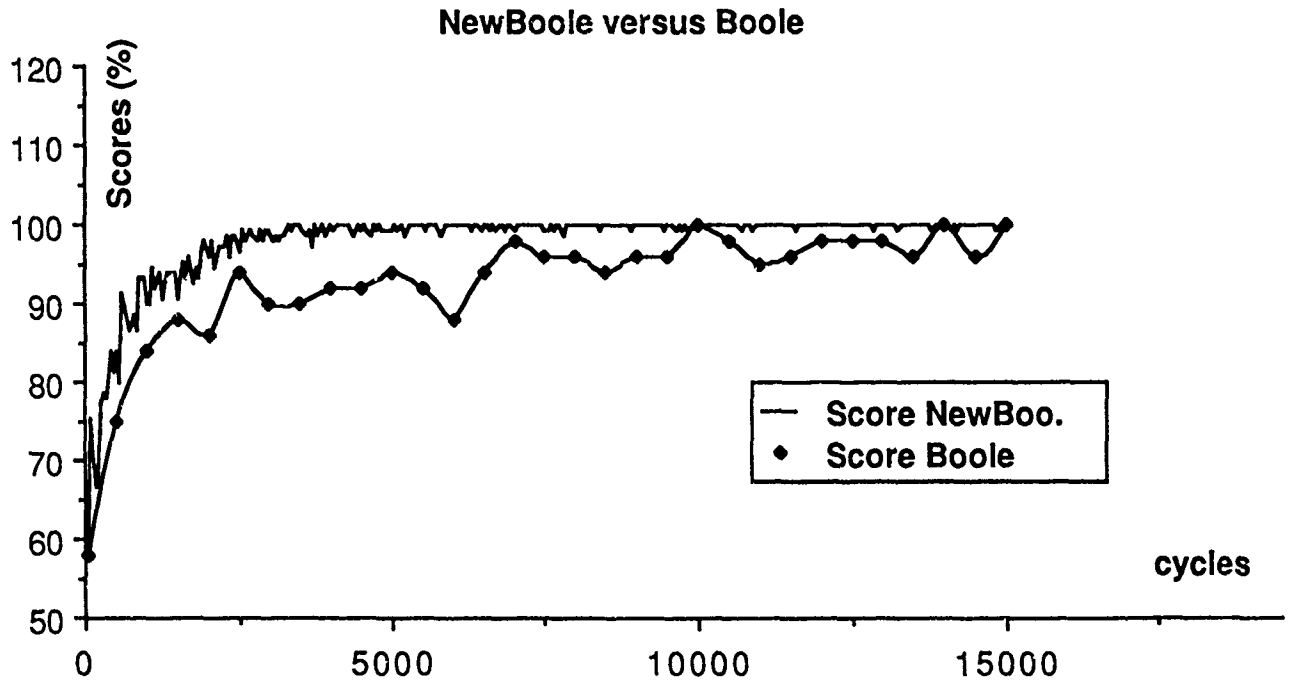


Figure 2: Comparison between untuned NewBoole and Boole.

Cycles	Score (%) Boole	Score (%) untuned NewBoole	Error(%) Boole	Error (%) NewBoole
0	48	50	52	50
500	75,9	86,7	24,1	13,3
1000	84,6	92,9	15,4	7,1
3200	91,2	99,8	8,8	0,2
5200	93,9	100	6,1	0
12000	97,3	100	2,7	0

Table 2: Comparison between Boole's and NewBoole's convergence rates

2/ We present in the second experiment (Figure 3) a "tuned" version of the NEWBOOLE algorithm.

* Firstly, we modified the values of certain parameters in order to speed up the learning process (see Table 1).

* Secondly, we modified the Performance Component in the following way: instead of selecting the "decision" classifier probabilistically, we systematically picked the highest ranked classifier in the match set: this deterministic selection affects in no way the learning process, since the Correct and NotCorrect sets are not determined in function of the selected classifier. This modification, as noted in [Booker, 89], permits a more steady convergence level; furthermore, since we are comparing NEWBOOLE with a deterministic algorithm (Back Propagation), it seemed logical to include some "determinism" in the algorithm.

We obtained an impressive learning rate after only 800 trials by modifying the value of the fraction deducted from the set of inaccurate classifiers ($p = 0.95$), the frequence of invocation of the genetic algorithm per cycle ($\rho = 4$), and the crossover rate ($\chi = 0.5$). This constitutes approximately a 17 fold improvement over Boole's original convergence rate.

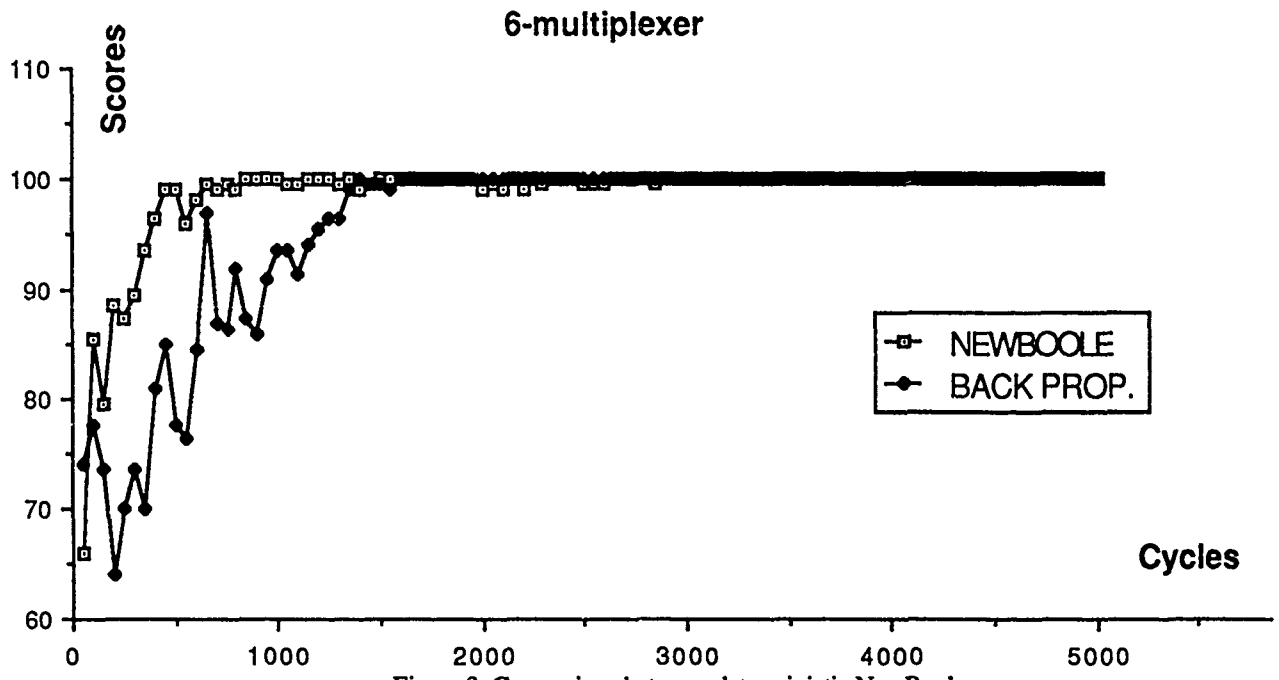


Figure 3: Comparison between deterministic NewBoole and Neural Net with BP on the 6 multiplexer problem

3/ Also in Figure 3, we present an experiment using a Neural Network with a Back-Propagation learning algorithm. The architecture (6:20-20-10-10:1)[1] and the parameters were tuned for this problem. Indeed, we see that convergence is reached after 1600 trials.

Of course, we noticed that a more complex network (6:100-50-40-30:1) converges within 900 trials. However, the performance is not better than with NEWBOOLE, and the memory occupied (6 x 100 + 100 x 50 + 50 x 40 + 40 x 30 + 30 = 8830 connections = 8830 x 4 = 35320 bytes) is unreasonably beyond what our population occupies (400 classifiers of 7 units each with 2 bits per unit = 800 bytes).

Therefore, when comparing NEWBOOLE with a Neural Net (NN) using BP, we restricted ourselves to reasonable networks.

[1]This notation means that the multilayered network has 6 inputs, then two layers of 20 cells, then two layers of 10 cells, and one output layer of one cell.

In all the networks, the parameters of each cell depend on the number of cell inputs N_{in}:
learning rate $\varepsilon = 0.1/\mathrm{sqrt}(N_{in})$, decay $\delta = 0$, noise $\theta = 0$, momentum $\alpha = 0$; weights $W_{ij}(0)$ are initialized randomly over the interval $[-1.5/N_{in}; 1,5/N_{in}]$.

Please also note that the simplest NN that can solve our problem (6:4:1) needs 7500 cycles to converge; this compares with the number of cycles NEWBOOLE needs to find the minimal set (around 5000 cycles for 80 % of minimal rules; the other rules have a very low strength and can easily be removed); however, NEWBOOLE finds this set automatically, whereas the NN architecture had to be provided at first.

4.2 NEWBOOLE and the 11-multiplexer problem.

Figure 4 shows the results obtained using NEWBOOLE to solve the 11-multiplexer compared with those obtained using the following neural net: (11:40-40-20-20:1).

The population was increased by a factor of 2,5 in order to fit the considerably larger classifier search space which grew by a factor of $3^{11} \times 2 / (3^6 \times 2) = 243$.

The number of links of the neural net rose much more (by a factor of 3260 / 730 = 4,5) than the population size.

Nonetheless, one notices that NEWBOOLE still converges at a faster rate than the neural net.

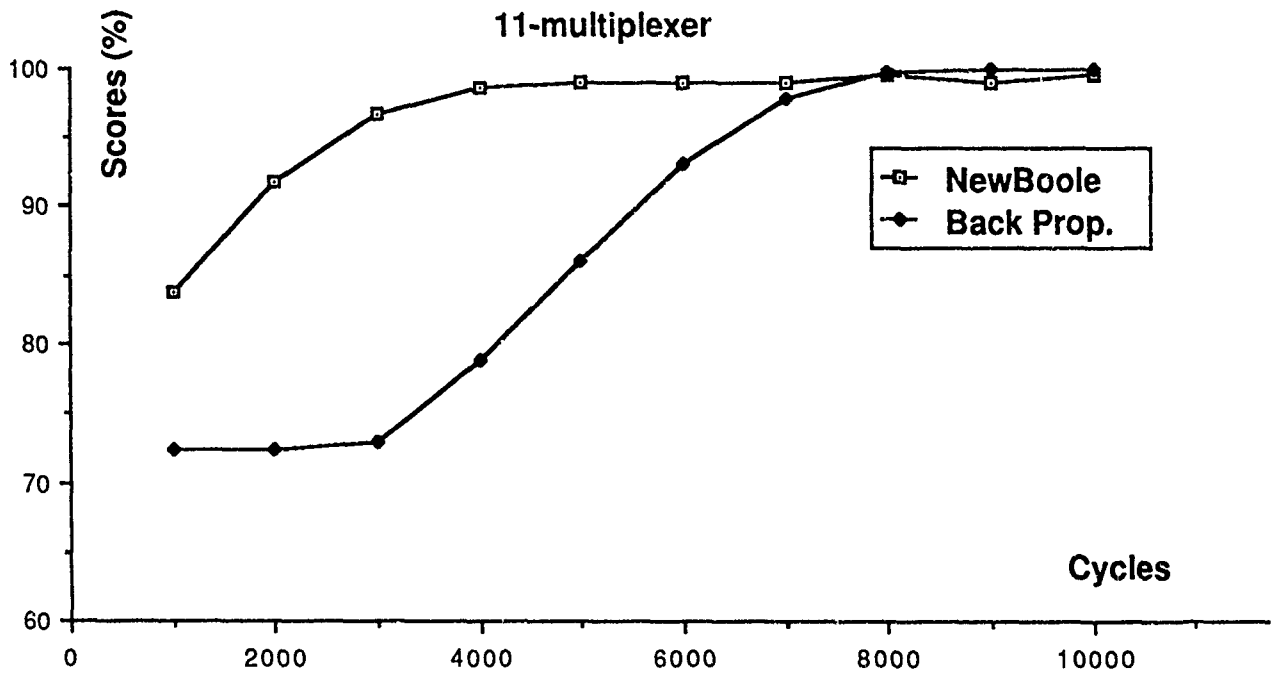


Figure 4: Comparison between deterministic NewBoole and Back-Propagation on the 11-multiplexer problem.

(symmetrically smoothed over 1000 cycles)

5. Conclusion

In this paper, we presented an improvement to the Boole system; indeed, we showed that convergence speed could be drastically improved, thus showing that GBML systems can learn far faster than were portrayed by Quinlan in [Quinlan, 88]. Furthermore, the comparison with an other incremental learning algorithm, the widely used neural net with back-propagation, showed that NEWBOOLE converges at least as fast. The basic difference between GBML and Connectionist learning thus seems to reside in the fact that Classifier Systems are rule based systems which provide comprehensive solutions, whereas neural networks merely provide sets of numerical coefficients without any semantic meaning.

6. References

[Booker, 89] "Triggered Rule Discovery in Classifier Systems", in ICGA 89, Morgan Kaufmann.

[Goldberg, 89] "Genetic Algorithms in search, Optimization, and Machine Learning", Addison Wesley.

[Holland, 75] "Adaptation in natural and artificial systems", Ann Arbor: University of Michigan Press.

[Holland 86] "Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems" In "Machine Learning: an Artificial Intelligence Approach, Vol 2, Michalski R.S., Carbonell J.G., Mitchell T.M. eds, Morgan Kaufmann, Los Altos (CA), 1986.

[Quinlan, 88], 'An Empirical Comparison of Genetic and Decision-Tree Classifiers", Proceedings of the fifth International Conference on Machine Learning.

[Wilson,87] "Classifier Systems and the Animat Problem", Machine Learning 2,199-228, 1987.

[Wilson & Goldberg, 89] "A critical Review of Classifier Systems", in ICGA 89, Morgan Kaufmann.

NEURAL NETWORK & REINFORCEMENT LEARNING

Learning Functions in k-DNF from Reinforcement

Leslie Pack Kaelbling*
Teleos Research
and
Stanford University

Abstract

An agent that must learn to act in the world by trial and error faces the *reinforcement learning* problem, which is quite different from standard concept learning. Although good algorithms exist for this problem in the general case, they are quite inefficient. One strategy is to find restricted classes of action strategies that can be learned more efficiently. This paper pursues that strategy by developing algorithms that can efficiently learn action maps that are expressible in k-DNF. Both connectionist and classical statistics-based algorithms are presented, then compared empirically on three test problems. Modifications and extensions that will allow the algorithms to work in more complex domains are also discussed.

1 Reinforcement Learning

Consider an agent that must learn to act in the world. At each moment in time, it gets information about the world from its sensors and must choose an action to take. Having executed an action, the agent gets a signal from the world that indicates how well the agent is performing; we shall call this a *reinforcement* signal. The reinforcement signal can be binary or real-valued and it will typically be noisy.

This learning scenario is quite different from standard concept learning, in which a teacher presents the learner with a set of input/output pairs. In the reinforcement-learning scenario, the agent must choose an output to generate in response to each input. The reinforcement signal it receives indicates only how successful that output was; it carries no information about how successful other outputs might have been. In addition, the fact that the reinforcement signal is noisy means that each output will have to be generated a number of times in order for the agent to acquire an accurate picture of which is better. In reinforcement-learning situations, an agent may choose an action because it expects it to have good results; however, it may also choose an action in order to gain information about its expected results. The tradeoff between acting to gain reinforcement and acting to gain information makes this problem especially interesting. The formal foundations of reinforcement learning have been widely studied [Kaelbling, 1989b, Kaelbling, 1989a, Narendra and Thathachar, 1989, Berry and Fristedt, 1985, Williams, 1986].

This paper will focus on a simple case of the reinforcement learning problem in which the following assumptions hold:

- the agent has only two possible actions
- the reinforcement signal at time $t+1$ reflects only the success of the action taken at time t
- reinforcement received for performing a particular action in a particular situation is 1 with some probability p and 0 with probability $1-p$ and each trial is independent
- the expected reinforcement value of doing a particular action in a particular input situation stays constant for the entire run of the learning algorithm

Section 6 discusses the extension of the results in this paper to situations in which each of the above assumptions is relaxed.

2 Complexity Versus Efficiency

There are a number of good algorithms for the reinforcement-learning scenario we are interested in, including learning-automata algorithms [Narendra and Thathachar, 1989], Sutton's reinforcement-comparison methods [Sutton, 1984], and Kaelbling's interval-estimation methods [Kaelbling, forthcoming]. These algorithms were originally developed for the case when the agent has no inputs other than reinforcement and merely needs to decide which action it should take all the time. They can be extended to the case of having many input situations simply by making a copy of the algorithm for each possible input situation. This method works well, but results in algorithms with space complexity proportional, at least, to the number of possible input situations. In addition, no generalization is exhibited, that is, the combined algorithms

*This work was supported by the Air Force Office of Scientific Research under contract F49620-89-C-0055.

do not take advantage of the common intuition that "similar" input situations are likely to require "similar" actions.

We can think of agents as learning *action maps*: mappings from input situations to actions. If an agent must be able to learn action maps of arbitrary complexity, then the methods described above are as good as any. However, if we restrict the class of action maps that we expect an agent to learn, we can invent algorithms for learning those maps that are much more efficient than algorithms for the general case.

A restriction that has proved useful to the concept-learning community is to the class of functions that can be expressed as propositional formulae in k-DNF. A formula is said to be in *disjunctive normal form* (DNF) if it is syntactically organized into a disjunction of purely conjunctive terms; there is a simple algorithmic method for converting any formula into DNF [Enderton, 1972]. A formula is in the class k-DNF if and only if its representation in DNF contains only conjunctive terms of length k or less. There is no restriction on the number of conjunctive terms—just their length. Whenever k is less than the number of atoms in the domain, the class k-DNF is a restriction on the class of functions.

Valiant was one of the first to consider the restriction to learning functions expressible in k-DNF [Valiant, 1984, Valiant, 1985]. He developed the following algorithm for learning functions in k-DNF from input-output pairs, which actually only uses the input-output pairs with output 0:

> *Let T be the set of conjunctive terms of length k over the set of atoms (corresponding to the input bits) and their negations and let L be the number of learning instances required to learn the concept to the desired accuracy.*[1]
>
> *for $i := 1$ to L do begin*
> *$v :=$ randomly drawn negative instance*
> *$T := T-$ any term that is satisfied by v*
> *end*
> *return T*

The algorithm returns the set of terms remaining in T, with the interpretation that their disjunction is the concept that was learned by the algorithm. This method simply examines a fixed number of negative instances and removes any term from T that would have caused one of the negative instances to be satisfied.[2]

The following sections describe algorithms for learning action maps in k-DNF from reinforcement and present the results of an empirical comparison of their performance. For each algorithm, the inputs are bit-vectors of length M, plus a distinguished reinforcement bit; the outputs are single bits.

3 Connectionist Methods for Learning k-DNF

There has been interesting work in the connectionist community on learning from reinforcement, which is relevant to our goals because it focuses on using more efficient algorithms to learn action maps in a restricted class of functions. This section will describe three connectionist methods: a linear reinforcement-comparison method, a multi-layer backpropagation method, and a hybrid method that combines Valiant's algorithm for concept learning with the linear reinforcement-comparison method.

These and other algorithms will be described in a standard form consisting of three components: s_0 is the initial internal state of the algorithm; $u(s,i,a,r)$ is the update function, which takes the state of the algorithm s, the last input i, the last action a, and the reinforcement value received r, and generates a new algorithm state; and $e(s,i)$ is the evaluation function, which takes an algorithm state s and an input i, and generates an action.

3.1 Linear Reward-Comparison Method

Most of the connectionist methods are simple single-layer algorithms that can learn action maps in the class of linearly separable functions [Widrow *et al.*, 1973, Sutton, 1984, Barto and Anandan, 1985]. Sutton [Sutton, 1984] performed extensive experiments on such methods and found that *reinforcement-comparison* algorithms tend to have the best performance. The equations below define Algorithm 8 from his dissertation [Sutton, 1984], which uses a version of the Widrow-Hoff or Adaline [Widrow and Hoff, 1960] weight-update algorithm.

> *The input is represented as an M-dimensional vector i. The internal state, s_0, consists of two M-dimensional vectors,* **v** *and* **w**.
>
> $$u(s,i,a,r) = \begin{array}{l} \textit{let } p := \sum_{j=1}^{M} v_j i_j \\ \textit{for } j = 1 \textit{ to } M \textit{ do begin} \\ \quad w_j := w_j + \alpha(r-p)(a-1/2)i_j \\ \quad v_j := v_j + \beta(r-p)i_j \\ \quad \textit{end} \end{array}$$
>
> $$e(s,i) = \begin{cases} 1 & \textit{if } \sum_{j=1}^{M} w_j i_j + \nu > 0 \\ 0 & \textit{otherwise} \end{cases}$$
>
> *where $\alpha > 0$, $0 < \beta < 1$, and ν is a normally distributed random variable of mean 0 and standard deviation δ_y.*

The output, $e(s,i)$, has value 1 or 0 depending on the inner product of **w** and i and the value of the random variable ν. The addition of the random value causes the algorithm to "experiment" by occasionally performing actions that it would not otherwise have

[1] This choice is not relevant to our reinforcement-learning scenario—the details are described in Valiant's papers [Valiant, 1984, Valiant, 1985].

[2] Valiant's presentation of the algorithm defines T to be the set of conjunctive terms of length k *or less* over the set of atoms and their negations; however, because any term of length less than k can be represented as a disjunction of terms of length k, we use a smaller set T for simplicity in exposition and slightly more efficient computation time.

taken. The updating of the vector w is somewhat complicated: each component is incremented by a value with four terms. The first term, α, is a constant that represents the learning rate. The next term, $r - p$, represents the difference between the actual reinforcement received and the predicted reinforcement, p. This serves to normalize the reinforcement values: the absolute value of the reinforcement signal is not as important as its value relative to the average reinforcement that the agent has been receiving. The predicted reinforcement, p, is generated using a standard linear associator that learns to associate input vectors with reinforcement values by setting the weights in vector v. The third term in the update function for w is $a - 1/2$: it has constant absolute value and the sign is used to encode which action was taken. The final term is i_j, which causes the jth component of the weight vector to be adjusted in proportion to the jth value of the input.

The space required for the state, as well as time for both update and evaluation operations is $O(M)$, where M is the number of input bits.

3.2 Multi-layer Back-propagation Method

Error back-propagation is a method for training connectionist networks that are comprised of multiple layers. Anderson [Anderson, 1986] has designed a connectionist system with multiple layers that uses backpropagation as a method for learning from reinforcement.

Anderson's system uses two networks: one for learning to predict reinforcement and one for learning which action to take. Each of these is a two-layer network, with all of the hidden units connected to all of the inputs and all of the inputs and hidden units connected to the outputs. The system was designed to work in worlds with delayed reinforcement (which are discussed here at greater length in Section 6), but it is easily modified to work in our simpler domain. This algorithm is rather complex, so space does not allow it to be described further. A clear description can be found in Anderson's dissertation [Anderson, 1986].

This method is theoretically able to learn very complex functions, but tends to require many training instances before it converges. The time and space complexity for this algorithm is O(MH), where M is the number of input bits and H is the number of hidden units.

3.3 A Hybrid Algorithm

Given our interest in restricted classes of functions, we can construct a new hybrid algorithm for learning action maps in k-DNF. It hinges on the simple observation that any such function can be expressed as a linear combination of terms in the set T, where T is the set of conjunctive terms of length k over the set of atoms (corresponding to the input bits) and their negations. It is possible to take the original M bit input signal and transduce it to a wider signal that is the result of evaluating each member of T on the original inputs. We can use this new signal as input to a relatively simple connectionist learning algorithm, such as the one described in Section 3.1 above.

If there are M input bits, the set T has size $C(2M, k)$ because we are choosing from the set of bits and their negations. However, we can eliminate all elements that contain both an atom and its negation, yielding a set of size $2^k C(M, k)$. The space required by the algorithm, as well as the time to update the internal state or to evaluate an input instance, is proportional to the size of T, and thus, $O(M^k)$. It is important to note that this algorithm (as well as the other three discussed in this paper) is *strictly incremental*: its time and space requirements depend only on the size of the input and on the fixed parameter k and do not increase over the course of a run.

4 Interval-Estimation Algorithm for k-DNF

The interval-estimation algorithm for k-DNF is, like the hybrid algorithm described in Section 3.3, based on Valiant's algorithm, but the interval-estimation algorithm uses standard statistical estimation methods rather than connectionist weight-adjustments. The technique of interval-estimation has also been applied to other reinforcement-learning problems [Kaelbling, forthcoming].

4.1 General Description

This section will describe the algorithm independent of particular statistical tests, which will be introduced in the next section. We shall need the following definitions, however. An input bit-vector *satisfies* a term whenever all the bits mentioned positively in the term have value 1 in the input and all the bits mentioned negatively in the term have value 0 in the input. The quantity $er(t, a)$ is the expected value of the reinforcement that the agent will gain, per trial, if it generates action a whenever term t is satisfied by the input and action $\neg a$ otherwise. The quantity $ubr_\alpha(t, a)$ is the upper bound of a $100(1-\alpha)\%$ confidence interval on the expected reinforcement gained from performing action a whenever term t is satisfied by the input. We can now give the formal definition of the algorithm:

s_0 = *the set T, with a collection of statistics associated with each member of the set*

$e(s, i)$ = *for each t in S*
if i satisfies t and
$ubr_\alpha(t, 1) > ubr_\alpha(t, 0)$ *and*
$\Pr(er(t, 1) = er(t, 0)) < \beta$
then return 1
return 0

$u(s, i, a, r)$ = *for each t in S*
update_term_statistics(t, i, a, r)
return s

At any moment in the operation of this algorithm, we can extract a symbolic description of its current hypothesis. It is the disjunction of all terms t such that $ubr_\alpha(t, 1) > ubr_\alpha(t, 0)$ and $\Pr(er(t, 1) = er(t, 0)) < \beta$. This is the k-DNF expression according to which the agent is choosing its actions.

The evaluation criterion is chosen in such a way as to make the important trade-off between acting to gain information and acting to gain reinforcement. A naive method would be for each term to generate a 1 whenever action 1 has had a higher success rate than action 0. This would be a very bad strategy, however, because if the first trial of action 0 failed, its success rate would be 0, causing action 0 never to be chosen again. The interval estimation method works because of the fact that the value of *ubr.* can be high for two reasons. It may be high because the confidence interval is very large due to the action not having been tried very often—this will cause the action to be chosen in order to gain information. The upper bound may also be high because the confidence interval is small and the action has a genuinely high payoff—this will cause an action to be chosen in order to gain reinforcement. At the beginning of a course of execution of this algorithm, actions are chosen almost at random, until the upper bound of the worse action is driven down by sampling, while the upper bound of the other stays high. The value of α determines the size of the confidence interval: when it is small the confidence interval is large and the algorithm is very conservative. It is not likely to converge to the wrong action, but it may take a long time to converge. As α is increased, the confidence intervals become smaller, the learning rate faster, and the chance of gross error higher.

Let the *equivalence probability* of a term be the probability that the expected reinforcement is the same no matter what choice of action is made when the term is satisfied. The second requirement for a term to cause a 1 to be emitted is that the equivalence probability be small. Without this criterion, terms for which no action is better will, roughly, alternate between choosing action 1 and action 0. Because the output of the entire algorithm will be 1 whenever any term has the value 1, this alternation of values can cause a large number of wrong answers. Thus, if we can convince ourselves that a term is irrelevant by showing that its choice of action makes no difference, we can safely ignore it.

4.2 Statistics

In the simple reinforcement-learning scenario we are considering, the necessary statistical tests are also quite simple. For each term, we store the following statistics: n_0, the number of trials of action 0; s_0, the number of successes of action 0; n_1, the number of trials of action 1; and s_1, the number of successes of action 1. These statistics are incremented only when the associated term is satisfied by the current input instance.

If n is the number of trials and s the number of successes arising from a series of Bernoulli trials with success probability p, the upper bound of a $100(1-\alpha)$ percent confidence interval for p can be approximated by

$$h(s,n,\alpha) = \frac{\frac{s}{n} + \frac{z^2_{\alpha/2}}{2n} + \frac{z_{\alpha/2}}{\sqrt{n}}\sqrt{\left(\frac{s}{n}\right)\left(1-\frac{s}{n}\right) + \frac{z^2_{\alpha/2}}{4n}}}{1+\frac{z^2_{\alpha/2}}{n}},$$

where $z_{\alpha/2}$ is such that $\Pr(Z \geq z_{\alpha/2}) = \Pr(Z \leq -z_{\alpha/2}) = \alpha/2$ when Z is a standard normal random variable [Larsen and Marx, 1986]. This allows us to define $ubr_\alpha(t,0)$ as $h(s_0,n_0,\alpha)$ and $ubr_\alpha(t,1)$ as $h(s_1,n_1,\alpha)$, where s_0, n_0, s_1, and n_1 are the statistics associated with term t.

To test for equality of the underlying Bernoulli parameters, we use a two-sided test at the β level of significance that rejects the hypothesis that the parameters are equal whenever

$$\frac{\frac{s_0}{n_0} - \frac{s_1}{n_1}}{\sqrt{\frac{\left(\frac{s_0+s_1}{n_0+n_1}\right)\left(1-\frac{s_0+s_1}{n_0+n_1}\right)(n_0+n_1)}{n_0 n_1}}} \text{ is either } \begin{cases} \leq -z_{\beta/2} \\ \text{or} \\ \geq +z_{\beta/2} \end{cases},$$

where $z_{\beta/2}$ is a standard normal deviate [Larsen and Marx, 1986]. Because sample size is important for this test, the algorithm is slightly modified to ensure that, at the beginning of a run, each action is chosen a minimum number of times, referred to by the parameter β_{min}.

The complexity of this algorithm is the same order as that of the hybrid connectionist algorithm of Section 3.3, namely $O(M^k)$.

5 Empirical Comparison

This section reports the results of a set of experiments designed to compare the performance of the algorithms discussed in this paper.

5.1 Algorithms and Environments

The following algorithms were tested in these experiments:

- LINCONN Linear reinforcement-comparison algorithm
- LINCONN+ Linear reinforcement-comparison with an extra input wired to have a constant value
- CONNKDNF Hybrid connectionist algorithm for k-DNF
- IEKDNF Interval-estimation algorithm for k-DNF
- BP Anderson's error back-propagation algorithm
- IE Basic interval-estimation algorithm

The basic interval-estimation algorithm IE [Kaelbling, forthcoming] is included as a yardstick, it is computationally much more complex than the other algorithms and will very likely out-perform them.

Each of the algorithms was tested in three different environments. The environments are called *binomial Boolean expression worlds* and can be characterized by the following parameters: M, *expr*, p_{1s}, p_{1n}, p_{0s}, and p_{0n}. The parameter M is the number of input bits, *expr* is a Boolean expression over the input bits, p_{1s} is the probability of receiving reinforcement value 1 given that action 1 is taken when the input instance satisfies *expr*; p_{1n} is the probability of receiving reinforcement value 1 given that action 1 is taken when the input instance does not satisfy *expr*; p_{0s} is the probability of receiving reinforcement value 1 given that action 0

Task	M	p_{1s}	p_{1n}	p_{0s}	p_{0n}
1	3	.6	.4	.4	.6
2	3	.9	.1	.1	.9
3	6	.9	.5	.6	.8

Table 1: Parameters of test environments for k-DNF experiments.

is taken when the input instance satisfies *expr*; and p_{0n} is the probability of receiving reinforcement value 1 given that action 0 is taken when the input instance does not satisfy *expr*. Input vectors are chosen by the world according to a uniform probability distribution.

Table 1 shows the values of these parameters for each task. The first task has the simple, linearly separable expression $(i_0 \wedge i_1) \vee (i_1 \wedge i_2)$; what makes it difficult is the small separation between the reinforcement probabilities. Task 2 has highly differentiated reinforcement probabilities, but the function to be learned, $(i_0 \wedge \neg i_1) \vee (i_1 \wedge \neg i_2) \vee (i_2 \wedge \neg i_0)$, is a complex exclusive-or. Finally, Task 3 is the simple conjunctive function, $i_2 \wedge \neg i_5$, but all of the reinforcement probabilities are high and there are 6 input bits rather than only 3.

5.2 Parameter Tuning

Each of the algorithms has a set of parameters. For both IEKDNF and CONNKDNF, $k = 2$. The simple connectionist algorithms LINCONN and LINCONN+ as well as CONNKDNF have parameters α, β, and σ. Following Sutton [Sutton, 1984], parameters β and σ in CONNKDNF, LINCONN, and LINCONN+ will be fixed to have values .1 and .3, respectively. The IEKDNF algorithm has two confidence-interval parameters, $z_{\alpha/2}$ and $z_{\beta/2}$, and a minimum age for the equality test β_{min}, while the IE algorithm has only $z_{\alpha/2}$. Finally, the BP algorithm has a large set of parameters: β, learning rate of the evaluation output units; β_h, learning rate of the evaluation hidden units; ρ, learning rate of the action output units; and ρ_h, learning rate of the action hidden units. In each of the tasks, the BP algorithm had as many hidden units as inputs.

All of the parameters for each algorithm were be chosen to optimize the behavior of that algorithm on the chosen task. The success of an algorithm was measured by the average reinforcement received per tick, averaged over the entire run. For each algorithm and environment, a series of 100 trials of length 3000 were run with different parameter values. Table 2 shows the best set of parameter values found for each algorithm-environment pair.

5.3 Results

Having chosen the best parameter values for each algorithm and environment, the performance of the algorithms was compared on runs of length 3000 using the parameter settings of Table 2. The performance metric was average reinforcement per tick, averaged over the entire run. The results are shown in Table 3, together with the expected reinforcement of executing a completely random behavior (choosing actions

ALG-TASK	1	2	3
LINCONN			
α	.0625	.125	.125
LINCONN+			
α	.125	.0625	.25
CONNKDNF			
α	.125	.25	.03125
IEKDNF			
$z_{\alpha/2}$	3	3.5	2.5
$z_{\beta/2}$	1	2.5	3.5
β_{min}	15	5	25
BP			
β	.1	.25	.1
β_h	.2	.3	.05
ρ	.15	.15	.35
ρ_h	.2	.05	.1
IE			
$z_{\alpha/2}$	3.0	1.5	2.5

Table 2: Best parameter values for each k-DNF algorithm in each environment.

ALG-TASK	1	2	3
LINCONN	.5329	.7418	.7769
LINCONN+	.5456	.7459	.7722
CONNKDNF	.5783	.8903	.7825
IEKDNF	.5789	.8900	.7993
BP	.5456	.7406	.7852
IE	.5827	.8966	.7872
random	.5000	.5000	.6750
optimal	.6000	.9000	.8250

Table 3: Average reinforcement for k-DNF problems over 100 runs of length 3000.

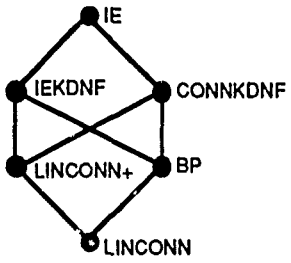


Figure 1: Significant dominance partial order among k-DNF algorithms for Task 1.

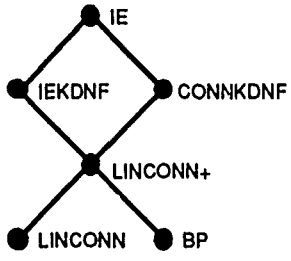


Figure 2: Significant dominance partial order among k-DNF algorithms for Task 2.

0 and 1 with equal probability) and of executing the optimal behavior. These results do not tell the entire story, however. It is important to test for statistical significance to be relatively sure that the ordering of one algorithm over another did not arise by chance. Figures 1, 2 and 3 show, for each task, a pictorial representation of the results of a 1-sided t-test applied to each pair of experimental results. The graphs encode a partial order of significant dominance, with solid lines representing significance at the .95 level and dashed lines representing significance at the .85 level.

With the best parameter values for each algorithm, it is also of some interest to compare the rate at which performance improves as a function of the number of training instances. Figures 4, 5, and 6 show superimposed plots of the learning curves for each of the algorithms. Each point represents the average reinforcement received over a sequence of 100 steps, averaged over 100 runs of length 3000.

5.4 Discussion

On Tasks 1 and 2 the basic interval-estimation algorithm, IE, performed significantly better than any of the other algorithms. The magnitude of its superiority, however, is not extremely great—Figures 4 and 5 reveal that the IEKDNF and CONNKDNF algorithms have similar performance characteristics both to each other and to IE. On these two tasks, the overall performance of IEKDNF and CONNKDNF were not found to be significantly different.

The backpropagation algorithm, BP, performed considerably worse than expected on Tasks 1 and 2. It is very difficult to tune the parameters for this algorithm, so its bad performance may be explained by

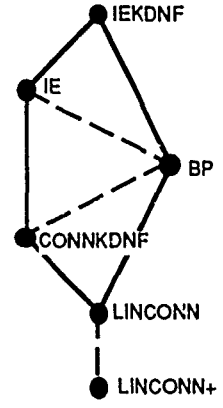


Figure 3: Significant dominance partial order among k-DNF algorithms for Task 3.

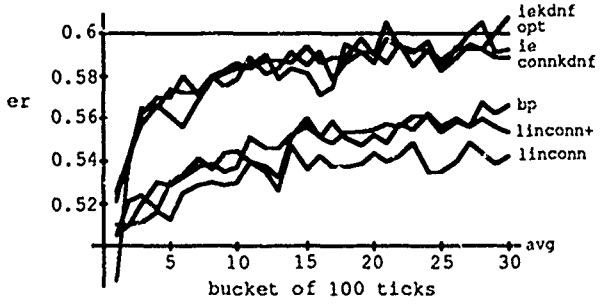


Figure 4: Learning curves for Task 1.

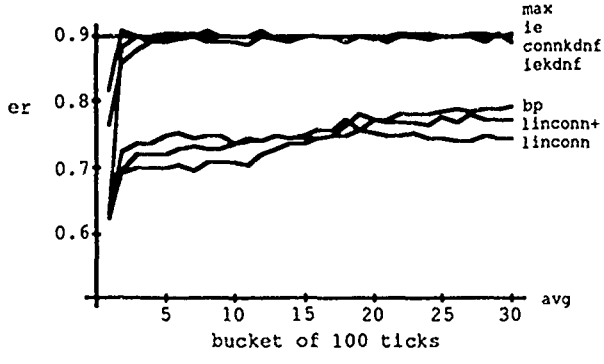


Figure 5: Learning curves for Task 2.

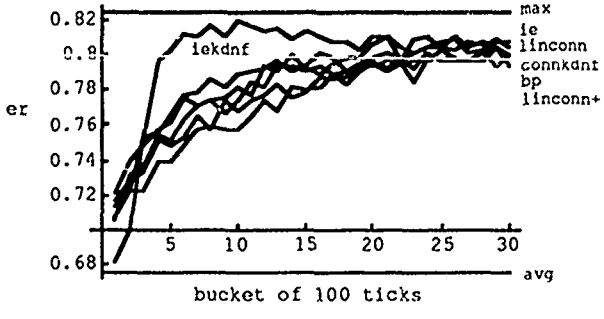


Figure 6: Learning curves for Task 3.

a sub-optimal setting of parameters.[3] However, it is possible to see in the learning curves of Figures 4 and 5 that the performance of BP was still increasing at the ends of the runs. This may indicate that with more training instances it would eventually converge to optimal performance.

The simple linear connectionist algorithms performed poorly on both Tasks 1 and 2. This poor performance was expected on Task 2, because such algorithms are known to be unable to learn non-linearly-separable functions. Task 1 is difficult for these algorithms because, during the execution of the algorithm, the evaluation function is often too complex to be learned by the simple linear associator. Adding a constant input value to the simple linear connectionist algorithm made a significant improvement in performance; this is not surprising, because it allows discrimination hyperplanes that do not pass through the origin of the space to be found.

Task 3 reveals many interesting strengths and weaknesses of the algorithms. One of the most interesting is that IE is no longer the best performer. Because the target function is simple and there is a larger number of input bits, the ability to generalize across input instances becomes important. The IEKDNF algorithm is able to find the correct hypothesis early during the run (this is apparent in the learning curve of Figure 6). However, because the reinforcement values are not highly differentiated and because the size of the set T is quite large, it begins to include extraneous terms due to statistical fluctuations in the environment, causing slightly degraded performance.

The IE, BP, and CONNKDNF algorithms all have very similar performance on Task 3, with the linear connectionist algorithms performing slightly worse, but still reasonably well.

6 Relaxing the Assumptions

This section will discuss the consequences of relaxing the assumptions made at the beginning of this paper, especially in the context of the two better-performing algorithms, IEKDNF and CONNKDNF. In some cases, simple changes can be made to the algorithms that will allow them to work in the more general situations. In others, there are theoretical problems that make extensions difficult. Each of the concrete extensions proposed to the IEKDNF algorithm has been implemented and tested.

Thus far we have assumed that the agent has only two possible actions. Many of the early learning-automata algorithms are directly applicable to problems with more than two actions. It has also been shown [Kaelbling, forthcoming] that the problem of generating actions specified by N output bits can be solved by N interconnected modules that learn to generate one output bit from reinforcement. Thus, the algorithms presented here could be applied, using this method, to problems with many possible outputs.

The problem of delayed reinforcement has been addressed by Sutton [Sutton, 1988] and Watkins [Watkins, 1989], among others. Sutton's solution, called the *temporal difference method* (TD) can be abstracted away from the particular reinforcement-learning mechanism being used. It provides a module that learns to transduce the delayed reinforcement signal that is coming from the world into an immediate reinforcement signal that evaluates each state of the world to be the expected future reward based on the agent's current strategy. Because this local reinforcement signal must be learned, using a TD module violates a different one of our assumptions: that the expected reinforcement of performing an action in a situation be fixed over the course of a run. This will be addressed below.

If the reinforcement provided by the world cannot be modeled as independent trials of some sort, then it is very difficult to use explicit statistical methods. The connectionist algorithms are implicitly statistical and would also have trouble in such worlds. However, if the trials are independent, we have a variety of different statistical models available. The CONNKDNF algorithm, as presented, can be used when the reinforcement is real-valued. The IEKDNF algorithm can be implemented with different statistical tests. For instance, if we know that the reinforcement values for each input-action pair are normally distributed, we can use standard statistical methods to construct confidence intervals and to test for equality of means. If we have no model, we can use non-parametric methods.

Finally, we consider the case of having the expected reinforcement of performing an action in a situation change during the course of a run. The CONNKDNF algorithm will work in such cases, although it might be necessary to adjust its parameters. The statistically-based IEKDNF algorithm can be modified to work, by causing its statistics to decay over time. If an action has not been tried for a long time, its n value will slowly decay, which will cause its confidence interval to grow larger. Eventually it will grow large enough for that action to be chosen again. If the action has good results, the policy will be changed to favor this action.

7 Conclusion

From this study, we can see that it is useful to design algorithms that are tailored to learning certain restricted classes of functions. The two specially-designed algorithms for out-performed standard methods of comparable complexity. The CONNKDNF and IEKDNF algorithms each have their strengths and weaknesses. It is possible that CONNKDNF may outperform IEKDNF to some extent because in CONNKDNF each term gets to contribute to the answer with different degrees. This avoids errors that occur in IEKDNF when a single term is barely over the threshold for gen-

[3] In the parameter tuning phase, the parameters were varied independently—it may well be necessary to perform gradient-ascent search in the parameter space, but that is a computationally difficult task, especially when the evaluation of any point in parameter space may have a high degree of noise.

erating a 1. On the other hand, the state of IEKDNF has internal semantics that are clear and directly interpretable in the language of classical statistics. This simplifies the process of extending the algorithm to apply to other types of worlds in a principled manner.

Important future work will be to identify other restricted classes of functions that can be learned efficiently and effectively from reinforcement and demonstrate that these classes contain functions that solve interesting and important problems from the real world.

Acknowledgments

Thanks to Stan Rosenschein for providing financial and moral support and to Rich Sutton for helpful discussions of connectionist and statistical reinforcement learning methods.

References

[Anderson, 1986] Charles W. Anderson. *Learning and Problem Solving with Multilayer Connectionist Systems.* PhD thesis, University of Massachusetts, Amherst, Massachusetts, 1986.

[Barto and Anandan, 1985] A. G. Barto and P. Anandan. Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:360–374, 1985.

[Berry and Fristedt, 1985] Donald A. Berry and Bert Fristedt. *Bandit Problems. Sequential Allocation of Experiments.* Chapman and Hall, London, 1985.

[Enderton, 1972] Herbert B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, New York, New York, 1972.

[Kaelbling, 1989a] Leslie Pack Kaelbling. A formal framework for learning in embedded systems. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 350–353, Ithaca, New York, 1989. Morgan Kaufmann.

[Kaelbling, 1989b] Leslie Pack Kaelbling. Foundations of learning in autonomous agents. In *Proceedings of the Workshop on Representation and Learning Autonomous Agents*, Lagos, Portugal, 1989.

[Kaelbling, forthcoming] Leslie Pack Kaelbling. *Learning in Embedded Systems.* PhD thesis, Stanford University, Stanford, California, forthcoming.

[Larsen and Marx, 1986] Richard J. Larsen and Morris L. Marx. *An Introduction to Mathematical Statistics and its Applications.* Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[Narendra and Thathachar, 1989] Kumpati Narendra and M. A. L. Thathachar. *Learning Automata: An Introduction.* Prentice-Hall, Englewood, New Jersey, 1989.

[Sutton, 1984] Richard S. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst, Massachusetts, 1984.

[Sutton, 1988] Richard S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[Valiant, 1984] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[Valiant, 1985] L. G. Valiant. Learning disjunctions of conjunctions. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 1, pages 560–566, Los Angeles, California, 1985. Morgan Kaufmann.

[Watkins, 1989] Christopher John Cornish Hellaby Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, 1989.

[Widrow and Hoff, 1960] Bernard Widrow and Marcian E. Hoff. Adaptive switching circuits. In *IRE WESCON Convention Record*, New York, New York, 1960. Reprinted in *Neurocomputing: Foundations of Research*, edited by James A. Anderson and Edward Rosenfeld, MIT Press, Cambridge, Massachusetts, 1988.

[Widrow *et al.*, 1973] Bernard Widrow, Narendra K. Gupta, and Sidhartha Maitra. Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3(5):455–465, 1973.

[Williams, 1986] Ronald J. Williams. Reinforcement learning in connectionist networks: A mathematical analysis. Technical report, Institute for Cognitive Science, University of California, San Diego, La Jolla, California, 1986.

Is Learning Rate a Good Performance Criterion for Learning?

Claude Sammut
Department of Computer Science
University of New South Wales
PO Box 1 Kensington
Australia 2033

James Cribb
Department of Computer Science
University of New South Wales
PO Box 1 Kensington
Australia 2033

Abstract

The most frequently used measure of performance for reinforcement learning algorithms is learning rate. That is, how many learning trials are required before the program is able to perform its task adequately. In this paper, we argue that this is not necessarily the best measure of performance and, in some cases, can even be misleading. In control tasks, such as pole balancing, we have found that a program that learns to balance the pole quickly produces a control strategy that is so specific as to make it impossible to transfer expertise from one related task to another. We examine the reasons for this and suggest ways of obtaining general control strategies. We also make the conjecture that, as a broad principle, there is a trade-off between rapid learning rate and the ability to generalise. We also introduce methods for analysing the results of reinforcement learning algorithms to produce readable control rules.

1. Introduction

The most frequently used measure of performance for reinforcement learning algorithms such as connectionist and genetic algorithms is learning rate. That is, how many learning trials are required before the program is able to perform its task adequately. In this paper, we argue that this is not necessarily the best measure of performance and, in some cases, can even be misleading. We illustrate our argument with a specific learning task, namely, pole balancing.

We are especially interesting in learning control tasks (or skill acquisition) and one of our primary goals is to be able to create a controller capable of dealing with a broad range of problems of the same type. In the course of performing experiments with a variety of learning algorithms (Sammut, 1988) we discovered that, while these algorithms could learn to balance a pole, the control strategy that had been acquired could not be frozen and transferred to another attempt at balancing a pole. In other words, the control strategy that had been learned was very specific. Furthermore, the more rapidly the program learnt to control the pole, the more specific was its control strategy. We will give reasons why this should be so and suggest ways of creating more general control strategies.

The first method, called "voting" requires the learning task to be repeated a number of times and the results of these learning runs are collected into a single controller that is very robust. We then examine how the results of this style of reinforcement learning can be turned into general and readable control rules.

All of the experiments described here use the BOXES algorithm (Michie and Chambers, 1968) as a starting point, however, we believe that our conclusions are applicable to most reinforcement learning algorithms, including connectionist and genetic algorithms. Indeed, we make the conjecture that, as a broad principle, there is a trade-off between rapid learning rate and the ability to generalise. This was characterised by Michie and Chambers as "exploration vs. exploitation".

In the following section we describe why rapid learning rates do not lead to general solutions. The following sections described a number experiments aimed at building more general controllers.

2. The Problem

The pole balancing problem will only be described very briefly since it has been the subject of a number previous papers and we are more interested in looking at the results of the learning algorithms and what to do

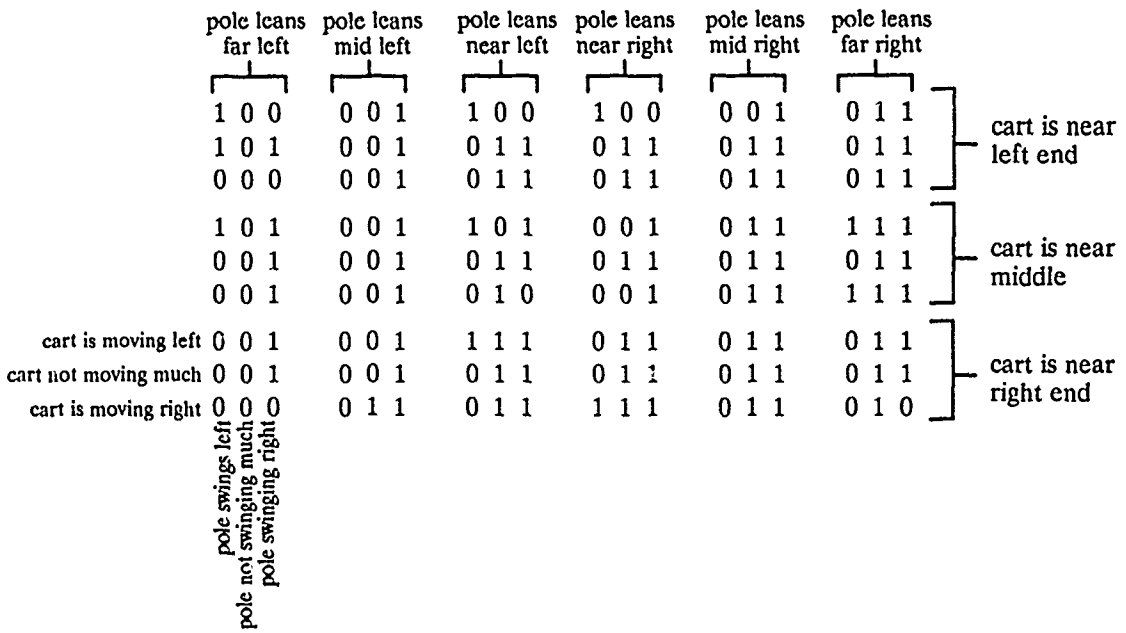


Figure 1: The 162 boxes displayed as a four-dimensional array

after they have done their job. Sammut (1988) gives a survey of previous work. The pole and cart problem can be stated as follows:

> A rigid pole is hinged to a cart, which is free to move within the limits of a track. The learning system attempts to keep the pole balanced and the cart within its limits by applying a force of fixed magnitude to the cart, either to the left or to the right. (Selfridge, Sutton and Barto, 1985)

Four parameters describe the state of the pole and cart at any instance in time. They are: the position of the cart on the track (x), the velocity of the cart ($\dot{x}$), the angle of the pole (θ), the angular velocity of the pole ($\dot{\theta}$). These parameters contain sufficient information to allow a controller to correctly decide whether to push left or right.

The problem space can be thought of as a four dimensional space, where each dimension is defined by one of the four state variables. Translating this to a computational structure creates a four dimensional look-up table. Given infinite storage, each point in the space contains a boolean value which tells the system to push left at that point or to push right. A practical approximation to this approach is to partition the state space so that neighbouring points within a region are mapped to the same decision (Michie and Chambers, 1968). In the experiments reported here, all the learning algorithms were tested against a simulation of the pole and cart. The simulation uses the equations of motion as described by Anderson (1987). We use the same partitions as Selfridge, Sutton and Barto (1985):

cart position:	$\pm$ 0.8, $\pm$ 2.4 metres
cart velocity:	$\pm$ 0.5, $\pm$ ∞ metres/sec
pole angle:	0, $\pm$ 1, $\pm$ 6, $\pm$ 12 degrees
angular velocity:	$\pm$ 50, $\pm$ ∞ degrees/sec

This creates 162 "boxes" that fill the problem space. After one time step in the simulation, the system will land in one box. The next move is determined by the action setting of that box i.e. push left or push right. We will display the action settings in the set of boxes as an array of zeros and ones as in Figure 1.

If the pole and cart system falls in a box containing a zero then the next control action is to push left. If it falls in a box containing a one then the control action is to push right. Other pole balancing algorithms need not represent the state space as explicitly as we do here. However, later we will argue that the conclusions drawn from these experiments are applicable to other systems.

Suppose we allow the program to learn how to control the pole and cart system by conducting a number of trials, i.e. attempts at balancing the pole, until it finally

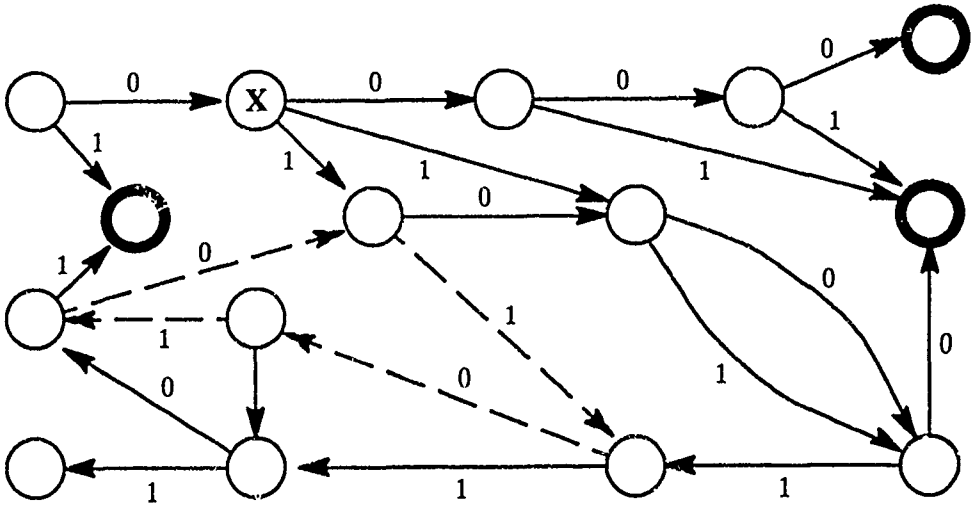


Figure 2: State transition representation of pole and cart problem

succeeds. Let us then freeze the learning element and use the control strategy that has been learnt on a new trial where the pole and cart are placed in a random, but recoverable, initial state. This can be repeated a number of times to obtain the average length of time that the controller is able to avoid failure. All of the learning algorithms tested by Sammut (1988), including BOXES (Michie and Chambers, 1968), the AHC algorithm (Sutton, 1984) and a genetic algorithm (Odetayo, 1988), could not successfully balance the pole again even for a short time. To see the reason for this, let us look at another way of representing the problem. Imagine that each box represents one node in a non-deterministic state transition diagram as in Figure 2. On entering a node, the program must chose whether to push left or rignt, i.e., exit the node on a zero transition or a one.

Where possible, the program should choose a transition that will put the system into a cycle since a cycle indicates that the pole and cart are avoiding the failure nodes (shown as heavy circles). The dotted lines in the diagram indicate a possible cycle. Note that choosing a 0 (left push) or 1 (right push) does not uniquely determine which node the system will enter since non-determinism has been introduced by approximating each point in the state space by a region or box. For example, the node marked 'X' in Figure 2 has two outgoing edges labelled '1'. If the system were to land in this node and the action in 'X' is a right push, then we cannot predict which of the two neighbouring nodes reachable by the '1' edges would be entered.

When a program learns to balance the pole quickly, that means that it has been able to rapidly find a cycle in which to keep the system and because the success criterion is keeping the pole balanced, the program does not explore other parts of the graph. Looking at the array of boxes in Figure 1, one can imagine that the program learns reliable settings for a small subset of boxes and if the pole and cart system can be kept within that reliable set of boxes then the pole will remain balanced. Since very little of the problem space has been explored, if the system is restarted with different initial parameters, then the program will be lacking in expertise and fail to keep the pole balanced.

While this effect can be observed using the BOXES representation, we claim that this also applies to other representations. For example, if a connectionist system is used and it has a number of hidden units and inputs directly from the four state variables, the learning program must explore a large space of settings for weights before settling on a solution. If the sole criterion for success is keeping the pole balanced just once, then very little of the space of weight combinations need be explored. Thus, as in other systems that learn by example, in order to generalise, the program must see a number of instances of a concept. However, here the instances are complete control strategies, in our case, represented by a set of boxes.

Selfridge, Sutton and Barto (1985) have shown that once the pole balancing task has been learned for one configuration of the system, learning algorithms can be created that will learn more rapidly how to control a

new configuration. So is it still desirable to be able to find a single control strategy? In a system that does not change too radically it would obviously be less wasteful not to have to re-learn each time a new control task arises. More importantly, in regular systems such as pole balancing (Makarovic, 1987) and satellite control (Sammut and Michie, 1989), it can be shown that very simple control rules can be effective. We wish to explore the possibility of learning these rules.

In the following section we give one method for deriving a robust control strategy.

3. Experiment 1: Voting

The apparent cause of the learning algorithm's "over specialisation" is that it converges too quickly on one strategy for balancing the pole and therefore does not try to explore more of the problem space. Our solution to this problem requires that we derive a number of these specialised control strategies and produce a generalisation of them. The method can be viewed as running a number of independent versions learning of the learning algorithm and averaging the outputs (cf. Buntine, 1989).

To test the reliability of a control strategy, we freeze the boxes and then use them to try to control the pole and cart from a new random starting position. This is repeated 20 times. The starting range is restricted to regions of the problem space from which it is possible to avoid failure, e.g. we do not start the pole learning so far over that it is impossible to swing it back towards vertical. If the pole can be balanced for 10,000 time steps in 20 out of 20 different runs then we deem the control strategy to be reliable. After one learning run whose success criterion is 10,000 time steps we usually get 0/20 performance on the reliability test. Even when the criterion is set at 100,000 we do little better. Thus, a single long learning run still does not guarantee good search of the state space. Details of all of the experiments described in this paper are given in Cribb (1989).

To collect information from a number of successful, but specific, control strategies we require each box to maintain a count of "votes" for each possible control action and at the end of a successful trial, the box casts a vote for the action to which it is currently set. Thus, after learning to balance the pole we scan each of the 162 boxes and if a box has an action setting of '0' then the '0' vote is incremented by 1 and similarly boxes that have an action setting of '1' have the '1' vote incremented. The program then starts the next trial as usual. In one experiment, after 832 trials, 100 trials had been successful, thus giving 100 rounds of voting. The majority vote from each box was extracted and used to create a new set of boxes that could be tested for reliability. That is, each box in the new set of boxes was obtained by comparing the '0' and '1' votes and if '0' had more votes then the box in the new set was set to '0' and similarly for boxes where '1' won the "election". Testing the new set of boxes resulted in 20/20 successful trials. Thus, voting holds some promise as a method for generalising control strategies.

How many elections are required before we can be sure of having a robust control strategy, if at all? To examine this question, we performed the reliability test after each round of voting. The results are plotted in Figure 3 where the vertical axis shows the average length of time the pole was balanced for in the 20 trials of the reliability test.

The graph suggests that it is necessary to learn to balance the pole on about 20 different occasions before we can construct a reliable control strategy using the BOXES algorithm. When different learning algorithms are used in conjunction with voting, the results may vary. In particular, we found that variants of BOXES that learn to the balance the pole more rapidly can take much longer to stabilise when used in voting.

4. Experiment 2: Coercion

Now that we have a way of producing a reliable control strategy, how do we extract simple and readable rules if they exist? One of the advantages of the boxes representation is that provides a simple "map" of the problem space that can be examined for regularities and we can use these regularities to simplify the controller. For example, the set of boxes shown in Figure 1 was obtained by having a program learn how to balance the pole 32 different times and then collecting the votes from each attempt. There are hints of regularity in the patterns of zeros and ones, e.g. the rightmost column consists entirely of ones except for the last box. There also appears to be some symmetry which we would expect in this problem. Perfectly clear patterns do not emerge because not all boxes are visited with equal frequency, thus some will have fewer opportunities to learn than others and will have settings that rely on statistically unsound information. Thus the set of boxes contains "noise" which we can try to clean up.

It is worth noting that in some regions of the state space (i.e. in some boxes) it doesn't really matter which action is chosen. If the pole is balanced and the cart is

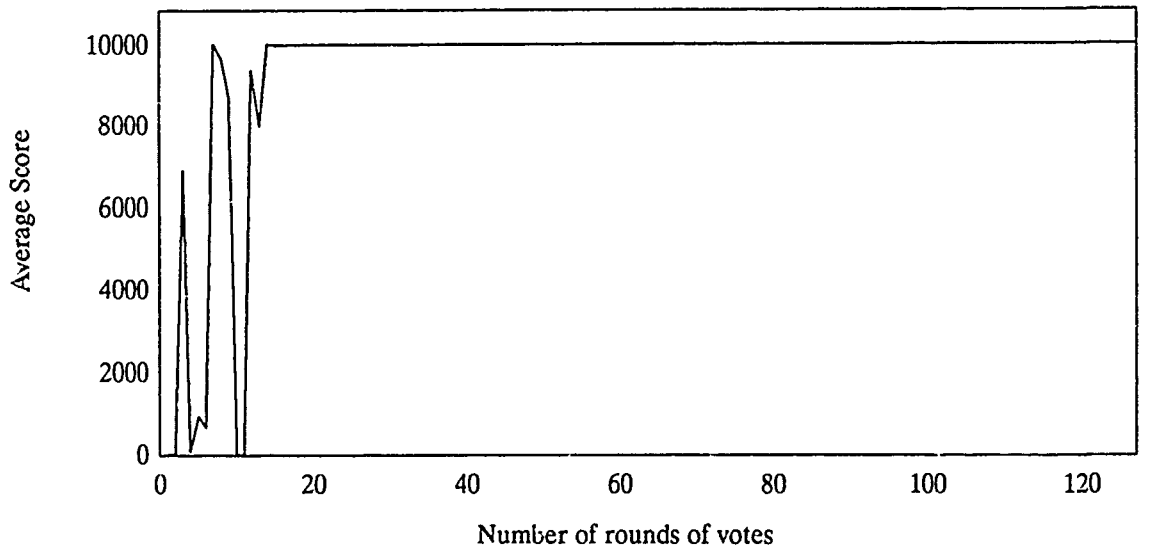


Figure 3: Scores after successive "elections"

near the middle if the track, or if the system is on the brink of failure and irrecoverable then whether the cart is pushed left or right makes very little difference. The votes for different control actions in these boxes are usually very close, giving an indication of their indecision. By applying a χ^2 test to the votes, we can identify those boxes that are indecisive and "coerce" the votes so that boxes conform with their surroundings. We use a nearest neighbour averaging method to choose the settings of indecisive boxes. In this way, we obtain the new set of boxes shown in Figure 4. It is apparent that the array now displays a symmetry that would be expected in this particular task.

```
001  001  001  001  001  011
001  001  001  001  011  011
001  001  001  001  011  011

001  001  001  001  011  011
001  001  001  011  011  011
001  001  011  011  011  011

001  001  011  011  011  011
001  001  011  011  011  011
001  011  011  011  011  011
```

Figure 4: "Coerced" boxes

After experimenting with this averaging method for cleaning up noise we began to realise that there are similarities between the this and image processing. The set of boxes can be thought of as forming a four-dimensional "retina" that contains an image of the world. Just as in image processing we wish to produce clean images and to detect patterns in the image. At this stage we have not pursued this analogy further but it appears to be worthwhile to investigate how other methods from image processing could be borrowed to assist us.

In the next section, we discuss how we can use the coerced boxes to simplify the representation of the state space and obtain control rules for the pole and cart problem.

5. Experiment 3: Merging

From Figure 4 it is clear that there is very little difference between the outer two major columns and the inner two. There is also some horizontal symmetry. This observation leads us to try to merge regions of the state space.

One of the limitations of the BOXES algorithm is that partitioning the state space must be done by hand, prior to any learning. If partitioning is too coarse, it may be impossible to learn the control task since a box may cover a region of the state space that is too large to class as requiring only a left push or a right push. If partitioning is very fine then it is more likely that a successful control strategy can be learnt but the time required increases with the number of boxes and the strategy also becomes difficult to express simply. For example, in our original set of boxes we effectively have 162 control

rules for a relatively simple task. This number can be reduced significantly if we merge boxes by eliminating and adjusting some of the partitions.

As an example of merging, let us examine the two left-most major columns in Figure 4, i.e. the "pole leans far left" and "pole leans mid left" columns. They differ in only one box, which we will call *B*. Any attempt to coerce this box renders the whole control strategy unreliable. At first, this appears to prevent us from merging the two columns, but it is possible.

The BOXES algorithm treats each box as an independent learning unit, i.e. individual boxes do not know what any other box has learnt. However, boxes do learn to operate in a cooperative manner because the scheme of rewards and punishments that the algorithm uses tells each box implicitly how well it is doing in relation to the whole system. Michie and Chambers (1968) describe this scheme in detail. In fact box *B* cannot be changed because it cooperates with other boxes in the centre columns. So if one box is to be changed, it may be necessary to change others as well. Since we do not know which boxes cooperate with each other, when we re-partition the state space, we also have to re-learn box settings.

We adopted the policy that only those boxes that did not pass the χ^2 test could be coerced. Box *B* did pass the test, i.e. the majority of attempts at learning to control the pole set this box to one. Therefore there is a genuine, small difference between the control strategies required for the two different regions. To account for this, when we merge the regions we shift the adjacent partitions. We define the merging operation as follows:

1. Two partitions are candidates for merging if the set of boxes contained in them are similar to a given degree, say 95%.

2. We can measure their degree of similarity by taking each box in turn in one of the regions, comparing its setting with that of the corresponding box in the other region and keeping a tally of the proportion of matches to mismatches.

3. If the tallies indicate that the two regions are sufficiently similar, we can eliminate the threshold that separates them and reset the adjacent thresholds, making their new values mid-way between their old values and the value of the eliminated threshold.

Having obtained the coerced set of 162 boxes in Figure 4 we proceeded to simplify the partitions. This can be done in a conservative way:

1. Merge only those partitions with the highest degree of similarity.
2. Learn how to control the system using the new partition.
3. Collect votes from at least 20 learning runs.
4. Coerce the indecisive boxes.
5. Repeat the whole process.

Steps 1 and 4 may have several candidates, not all of which are reliable according the the criterion described earlier. Therefore one candidate is selected by running the reliability test.

By iterating through this procedure we obtained a set of 108 boxes, then 72 and finally the 54 boxes shown in Figure 5.

```
001  001  011
001  001  011
001  011  011

001  001  011
001  011  011
001  011  011
```

Figure 5: Simplified set of boxes after merging

The representation is now simple enough that we can read off a set of rules in the form of an if-then-else statement.

```
if       θ̇ < -5    then left
else if  θ̇ > 5     then right
else if  θ < -2    then left
else if  θ > 2     then right
else if  ẋ < -0.1  then left
else if  ẋ > 0.1   then right
else if  x < 0     then left
else if  x > 0     then right
```

This can be restated simply: "If the pole is swinging rapidly or leaning well over in one direction then push in that direction. If the pole is balanced but the cart is over to one side then push toward that side". The second part of the rule is interesting because it seems counter-intuitive at first but in order to move the pole and cart toward the middle of the track, the pole should first be leaning in the direction we in which we want to go so the first step is to move in the opposite direction to swing the pole in the correct direction.

At this point we should also note a further analogy with region finding in image processing. Region finding algorithms attempt to either merge or split neighbouring regions in an attempt to find a meaningful structure in the image. In our case we are merging regions only, however, in discussing future work Michie and Chambers (1968) suggested that automatically merging and splitting boxes was desirable. Our work goes some way toward this goal but further work is required, particularly to obtain a data structure for boxes that can makes splitting and merging simple.

6. Experimental Variations and Future Experiments

Cribb (1989) has performed an extensive array of experiments other than those we are able to report here. These include variations on the original BOXES algorithm and different methods for voting and coercion. Some of these experiments are summarised below and we also discuss further experiments that we intend to carry out.

In Figure 3 we showed how combining results from independent learning sessions allowed us to produce more reliable control strategies. This graph was derived from experiments where sets of boxes were combined and later coerced. We also attempted to coerce boxes "on the fly" and use the coerced boxes in voting. This method did not produce a reliable control strategy as consistently as the original method. It appears that coercion is only worth doing on sets of boxes that have already been found to be reliable.

The experiments described in this paper have been repeated on a pole and cart system where the forces applied to the cart were asymmetrical. That is, the force applied when pushing to the left was 10 Newtons while the force used to push to the right was only 5 Newtons. This is an inherently less stable system and therefore takes longer to learn to control. The asymmetrical nature of the problem is also parallelled by asymmetry in the boxes.

We mentioned earlier that the BOXES algorithm has a systems of rewards and punishments that implicitly forces boxes to cooperate with each other. When the problem is asymmetrical it appears that this global knowledge is less useful. A variation of the learning algorithm that only used local knowledge of boxes on average converged more quickly than the original algorithm. However, when this algorithm was used in conjunction with voting, many more elections were required before we could not guarantee that a reliable set of boxes would result. Thus we need to investigate much more thoroughly the conditions under which the voting method can be used with confidence. This experiment also supported our view that rapid convergence is not always consistent with generality.

To properly validate our approach it will be necessary to repeat these experiments using other control tasks. We have begun preliminary work on learning to control a double pole, that is, one pole is balanced on the end of another as shown in Figure 6. This gives us a 6-dimensional state space. This task is useful because with six dimensions it is impossible to visually inspect boxes for regularities and thus eliminate any possibility of bias on our part when writing our algorithms.

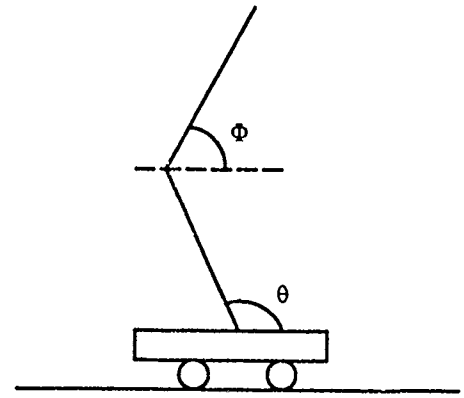


Figure 6: The double pole and cart

We have also used a satellite control problem in some experiments. Sammut and Michie (1989) give a full account of this work. We adapted the pole balancing rules above so that they could control the thrusters of a satellite, the goal being to maintain a stable attitude for the satellite. This domain is somewhat more complicated than pole balancing because there are many more actions. In the case of the pole balancer there are only two possible actions: push left or push right. To control a satellite we can apply positive or negative torques for either roll, pitch or yaw. Thus there is a minimum of six actions. Furthermore, bang-bang control is not acceptable since propellant usage is critical for the longevity of the space craft. As the number of control actions increases, the number experiments required to find an appropriate box settings increases dramatically.

Sammut and Michie (1989) used the experience gained from learning pole balancing control strategies to manually construct a controller for the simulation of

a commercial satellite. The control rules are best expressed as a decision table, which happens to correspond closely to the boxes representation. The strongest similarity between pole balancing and satellite control is that we compare the rate of change and value of each variable with a threshold. One of the differences is that each of the variables roll, pitch and yaw can be treated independently. Thus each variable has its own decision table. Table 1 shows the control strategy for yaw only. To illustrate how the table works, when the yaw is positive but the yaw rate is very negative then the satellite fires its yaw thruster using one quarter of full thrust. Note that we are no longer using bang-bang control. Instead we use discrete steps in thrust: no thrust (0) one quarter of full thrust (1/4), one half (1/2) and full thrust (1). There a three such tables for each of roll pitch and yaw. At each time interval in the simulation, all three table are consulted so that three actions may be enabled simultaneously. We have not yet attempted to learn control strategies in this domain but intend to do so since the additional computational demands brought about by the complexity of the task require us to find more efficient methods of learning.

Table 1: Yaw Decision Table

Yaw positive	1/4	0	-1/4	-1/2	-1
Yaw ok	1/2	0	0	0	-1/2
Yaw negative	1	1/2	1/4	0	-1/4
	Yaw rate very negative	Yaw rate negative	Yaw rate ok	Yaw rate positive	Yaw rate very positive

7. Discussion

One of the most exciting results of this work has been the convergence of the experimentally derived rules shown above and rules that Makarovic (1987) obtained from a mathematical analysis of the equations of motion of the pole cart. Except for small differences in threshold values, the rules are the same. Moreover, we were able to transfer much of what we had learnt from pole balancing to other control tasks (Sammut and Michie, 1989). One important implication of these results is that, with further work, it may be possible to use these methods to learn the behaviour of 'black boxes' and then by analysing the behaviour, build a more abstract theory of the operation of the black box. That is, we see this style of learning as an important adjunct to scientific discovery programs.

There is still much that needs to be done in order to refine these methods into usable tools. We have mostly worked "by the seats of our pants", trying something when it looked right. This is unsatisfactory and requires a better theoretical understanding of the learning algorithms and further experimental work to validate some of our conjectures.

The goal of this work has been to use a reinforcement learning algorithm to acquire expertise in controlling a dynamic system and then analysing the results to produce simple, readable control rules. In the course of this analysis, we found that fast learning algorithms tend to produce controllers that are very specific to the task, i.e. they do not generalise well. In some respects this is similar to the tendency of some learning algorithms to over-fit data (Quinlan, 1987; Fisher, 1989).

Acknowledgments

Some of the work reported here was done while the first author was on study leave at the Turing Institute, Glasgow. Donald Michie was not only responsible for the original pole balancing experiments but made significant contributions to the work described here. We also thank Michael Bain for his assistance and Wray Buntine for some lively discussions.

References

Buntine, W. (1989). Learning Classification Rules Using Bayes. *Proceedings of the Sixth International Workshop on Machine Learning*. Ithaca NY. Morgan Kaufmann, pp. 94-98.

Cribb, J. (1989). Comparison and Analysis of Algorithms for Reinforcement Learning. Department of Computer Science, University of New South Wales.

Anderson, C.W. (1987). Strategy Learning with Multilayer Connectionist Representations. In Pat Langley (Ed.), *Proceedings of the Fourth International Workshop on Machine Learning. Irvine*, CA: Morgan Kaufmann.

Fisher, D.H. (1989). Noise Tolerant Conceptual Clustering. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit: Morgan Kaufamnn, pp 825–830.

Makarovic, A, (1987). Pole Balancing as a Benchmark Problem for Qualitative Modelling, DP–4953, Josef Stefan Institute, Ljubljana.

Michie, D. and Chambers, R.A. (1968). BOXES: An Experiment in Adaptive Control. In E. Dale and D. Michie (Eds.), *Machine Intelligence 2*. Edinburgh: Oliver and Boyd.

Odetayo, M. (1988). Balancing a Pole-Cart System Using Genetic Algorithms, Department of Computer Science, University of Strathclyde, Glasgow.

Quinlan, J.R. (1987). Simplifying Decision Trees, *International Journal of Man-Machine Studies*, **27**, pp 221–234.

Sammut, C. (1988). Experimental Results from an Evaluation of Algorithms that Learn to Control Dynamic Systems, In John Laird (Ed.), *Proceedings of the Fifth International Conference on Machine Learning*. Ann Arbor, Michigan: Morgan Kaufmann.

Sammut, C. and Michie, D. (1989). Controlling a 'Black Box' Simulation of a Space Craft, Turing Institute, Glasgow.

Selfridge, O.G., Sutton R.S. and Barto, A.G. (1985). Training and Tracking in Robotics. In *Proceedings of the Ninth International Conference on Artificial Intelligence*. Los Altos, CA: Morgan Kaufmann.

Sutton, R.S. (1984). Temporal Credit Assignment in Reinforcement Learning, COINS Technical Report 84–02, Department of Computer and Information Science, University of Massachusetts, Amherst.

Active Perception and Reinforcement Learning

Steven D. Whitehead and Dana H. Ballard*
Dept. of Computer Science
University of Rochester
Rochester, NY 114627

Abstract

This paper considers adaptive control architectures that integrate active sensory-motor systems with decision systems based on reinforcement learning. One unavoidable consequence of active perception is that the agent's internal representation often confounds external world states. We call this phenomenon *perceptual aliasing* and show that it destabilizes existing reinforcement learning algorithms with respect to the optimal decision policy. A new decision system that overcomes these difficulties is described. The system incorporates a perceptual subcycle within the overall decision cycle and uses a modified learning algorithm to suppress the effects of perceptual aliasing. The result is a control architecture that learns not only how to solve a task but also where to focus its attention in order to collect necessary sensory information.

1 Introduction

Recently there has been a resurgence of interest in intelligent control architectures that are based on reinforcement learning methods (RLM) [Barto *et al.*, 1989; Clocksin and Moore, 1988; Holland, 1986; Miller *et al.*, 1990; Sutton, 1988; Watkins, 1989; Whitehead and Ballard, 1989a; Wilson, 1987a]. These architectures are appealing because they are both reactive and adaptive. Unlike traditional plan based controllers, RLM systems do not make decisions by appealing to time consuming search through a space of possible plans. Instead, they maintain a *policy function* that maps situations directly into actions. Decision making reduces to computing the instantaneous value of the policy function and can be performed in constant time — e.g. a policy function can be implemented using a table, CMAC, or neural net (all of which can be evaluated in constant time).

*This work was supported in part by NSF research grant no. DCR-8602958, in part by NSF research grant no. DCR-8602958, and in part by NSF research grant no. IRI-8903582.

The immediacy of decision making puts RLM systems in close relationship with other reactive systems [Agre and Chapman, 1987; Brooks, 1986; Drummond, 1989; Firby, 1987; Georgeff and Lansky, 1987; Nilsson, 1989; Schoppers, 1987]. However, RLM systems distinguish themselves from these and most reactive systems in that they are *adaptive*. The vast majority of reactive systems *do not* learn. Instead, their decision knowledge is hand coded into them by their designers; either explicitly (e.g. [Agre, 1988; Brooks, 1986; Georgeff and Lansky, 1987; Firby, 1987]) or through hand coded causal models which eventually get compiled into a set of reactive rules (e.g. [Blythe and Mitchell, 1989; Fikes *et al.*, 1972; Laird *et al.*, 1986; Schoppers, 1989]). RLM systems do not rely on hand coded decision knowledge. They learn the optimal control strategy by interacting with the world and receiving feedback in the form of reinforcement. This adaptability relieves the burden of providing complete domain knowledge *a priori*, since it is acquired with experience. It also allows the system to adapt to changing circumstances and learn new tasks.

Although RLM systems are promising, to date they have only been applied to relatively simple tasks, such as pole balancing [Barto *et al.*, 1983; Sutton, 1984], simplified navigation [Barto and Sutton, 1981; Booker, 1982; Sutton, 1990; Watkins, 1989; Wilson, 1987a], and easy manipulation games [Anderson, 1989; Whitehead and Ballard, 1989a]. Before these systems can be scaled to larger, more complex control problems a number of issues must be addressed. These include developing techniques for improving the learning rate, developing more sophisticated uses for internally held goals, and incorporating more realistic models of perception and action. Early progress on the first two of these issues looks promising (for faster learning see [Sutton, 1990; Whitehead and Ballard, 1989a; Whitehead and Ballard, 1989b]; for models using internal goals see [Watkins, 1989; Whitehead, 1989; Wilson, 1987b].) This paper deals with the third issue of integrating realistic sensory-motor systems into RLM architectures.

The vast majority of work in AI has not dealt realistically with perception, and research in reinforcement learning is no exception. A common simplifying assumption is that a decoupled perceptual system au-

tomatically supplies a central control system with a database of logically consistent inputs that describes in detail each object in the domain — that is a set of propositions that describe the relationships between, and the features of all the objects in the domain. Unfortunately, even for simple toy domains this leads to large, complex internal representations and unrealistic assumptions about the capabilities of the perceptual system. For example, in a classical blocks world domain containing n blocks, the size of the state space using a traditional representation is $O(n!)$. For $n = 19$ that is over two trillion (2,147,483,647) states. Most of the information that distinguishes states in the internal representation is irrelevant to immediate task faced by the agent and only interferes with decision making (and learning) by clogging the system with irrelevant details. Further, these overly descriptive representations put undue pressure on the perceptual system to maintain their fidelity.

Agre and Chapman have recognized this problem and suggested *indexical representations*, a much more feasible approach based on active sensory-motor systems [Agre, 1988; Agre and Chapman, 1987; Chapman, 1989]. A central premise underlying indexical representations is that a system needn't name and describe every object in the domain, but instead should only register information about objects that are relevant to the task at hand. Further, those objects should be indexed according to the function they play in the current behavior. Two important implications of this approach are: 1) it leads to compact and limited scope input representations since at any moment the sensory system registers only the features of a few key objects; and 2) it leads to systems that actively control their perceptual apparatus — i.e. actively manipulate the binding between objects in the world and internal representational structures. Thus, in the case of a blocks world problem, the system would represent blocks immediately relevant to the task, and would be oblivious to the rest. As a result, the size of the internal representation reflects the complexity of the task being solved and not the number of objects in the domain (which could be arbitrary!).

We have been experimenting with systems that incorporate both indexical representations (for feasible perception) and RLMs (for adaptive control). In particular, we show that integrating indexical representations (and active perception, in general) and RLM into a single control architecture is non-trivial because the use of indexical representations results in internal states that are ambiguous with respect to the utility of the external world. We term this phenomenon *perceptual aliasing* and show that it leads to undesired local maxima in the decision system's evaluation function. These local maxima severely interfere with the decision system's ability to learn an adequate control policy. To overcome these difficulties a new RLM decision system is introduced that embeds a perceptual cycle within the overall decision cycle and uses a modified learning algorithm to eliminate the undesired local maxima. The result is a reactive architecture that learns both the overt physical action needed to solve a problem, and where to focus its attention in order to disambiguate the current situation with respect to the task. These ideas are illustrated in a system that learns a simple block manipulation task.

2 Foundations

2.1 Embedded Learning Systems

Before getting into the details of indexical representations, reinforcement learning, and perceptual ambiguity, it is useful to formalize concepts such as "the world", "the agent", "the sensory-motor system" and "the decision system". For this purpose we begin by adopting a formal model for describing embedded learning systems. The model is shown in Figure 1, and extends a model proposed by Kaelbling [Kaelbling, 1989] by representing the relationship between external world states and the agents internal representation.

The world is modeled as a deterministic automaton whose state changes depend on the actions of an agent. The world is formally described by the triple $(S_E, A_E, \mathcal{W})$, where S_E is the set of possible world states, A_E is the set of possible physical actions executable by the agent, and $\mathcal{W}$ is the state transition function mapping $S_E \times A_E$ into S_E. Our model of the agent is slightly more complex consisting of three components: a sensory-motor subsystem, a reward center, and a decision subsystem.

The sensory-motor subsystem implements three functions: 1) a perceptual function $\mathcal{P}$, 2) an internal configuration function $\mathcal{I}$, and 3) a motor function $\mathcal{M}$. Its main purpose is to relativize the external transductory signals S_E and A_E. On the sensory side the system transduces the world state into the agent's internal representation. Since perception is active, this mapping is dynamic and depends on the configuration of the sensory-motor apparatus. Formally, the relationship between external world states and the agent's internal representation is modeled by the *perceptual function* $\mathcal{P}$ which maps world states S_E and sensory-motor configurations C onto internal representations S_I (i.e. $\mathcal{P} : S_E \times C \rightarrow S_I$). On the motor side, the agent has a set of *internal* motor commands A_I that affect the model in two ways. they can either change the state of the external world (by being translated into external actions, A_E), or they can change the configuration of the sensory-motor subsystem. As with perception, the configuration of the sensory-motor subsystem modulates the effects of internal commands. This dependence is modeled by the functions $\mathcal{M}$ and $\mathcal{I}$ which map internal commands and sensory-motor configurations into actions in the external world and new sensory-motor configurations respectively. That is, $\mathcal{M} : A_I \times C \rightarrow A_E$ and $\mathcal{I} : A_I \times C \rightarrow C$.

The second subsystem in our model is the reward center. It implements a reward function $\mathcal{R}$, which maps external world states S_E into real valued rewards $\Re$. Rewards indicate that the world in a desirable state and are used by the decision subsystem to improve performance.

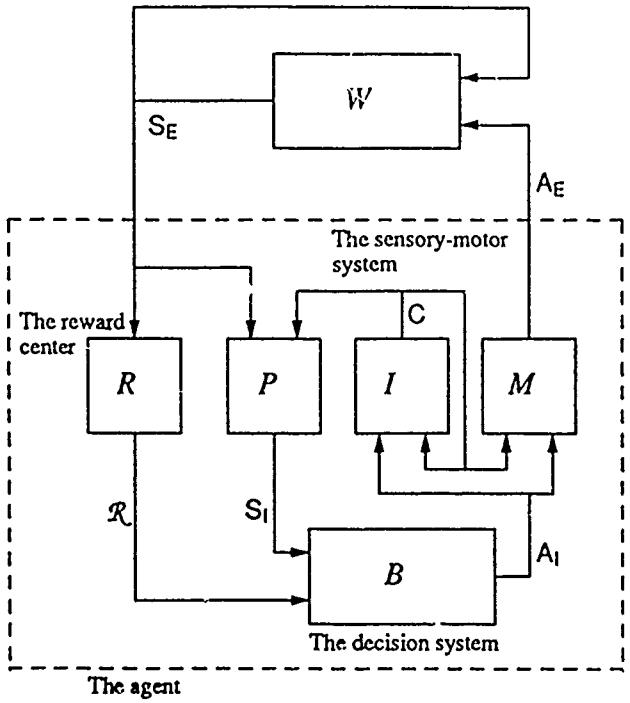


Figure 1: A formal model for embedded learning systems with active sensory-motor subsystems.

The final component of the agent is the decision subsystem. This subsystem is like a homunculus that sits inside the agent's head and controls its actions. On the sensory side, the decision subsystem does not have access to the state of the external world, but only the agent's internal representation. Similarly on the motor side, the decision subsystem generates internal action commands that are interpreted by the sensory-motor system. Formally, the decision subsystem implements a behavior function $\mathcal{B}$ mapping sequences of internal states and rewards, $(S_I \times \Re)^*$ into internal actions, A_I. The objective of the subsystem is to implement a control policy that maximizes its *return*, which is defined as a discounted sum of the reward received over time.

$$return = \sum_{n=0}^{\infty} \gamma^n r_{t+n} \qquad (1)$$

where r_t is the reward received at time t, and γ is a discount factor between 0 and 1.

2.2 Indexical Representations

The central concept underlying indexical representations is the *marker*, around which nearly all action and perception revolves. Markers are similar to variables, implemented by the sensory-motor system; they get dynamically bound to objects in the world and remain bound to those objects until being bound to other objects. Changing a marker's binding is accomplished by executing explicit actions specifically targeted for that marker. For example, a system might have a marker M_1 and an associated action *Warp-M_1-to-Red*, which is used to index and bind red objects to M_1. In this case, executing *Warp-M1-to-Red* causes the sensory-motor system to search the world for a red object and bind it to *M1*. If a red object cannot be found, the action fails and *M1*'s binding remains unchanged. If multiple red objects exist, the sensory-motor system chooses the first one it comes to. Markers play an important role in motor control since overt actions are predominately specified with respect to markers. In this case, a marker's binding acts to establish the reference frame in which the action is performed. For example, the overt action *Place-at-M1* might cause a robot to place an object it is holding at the location currently associated with marker *M1*. We distinguish two types of markers, overt markers and perceptual markers. A marker is overt if it has an action associated with it that affects the state of the external world. Otherwise it is a perceptual marker. Overt markers are used for establishing reference frames for action in the world, while perceptual markers are used for collecting additional information about the current state. Actions associated with overt markers are called *overt actions* and actions associated with perceptual markers are called *perceptual actions.*

A key feature of markers is the constraint that there are only a limited number of them, say less than ten (the system described below has two markers). The small number of markers and the limited number of features associated with each marker keeps both the internal representation and the number of possible actions much smaller than is possible with conventional representations. If an object in the world is not bound to a marker, then it is invisible to the system (except for the effects it registers in peripheral inputs). [1]

In an indexical representation the system's sensory inputs fall into three general categories, peripheral aspects, local aspects, relational aspects. *Peripheral aspects* register general, spatially non-specific information about the world, such as the presence or absence of certain colors, shapes, and motions. Both local aspects and relational aspects register properties of marked objects. *Local aspects* register intrinsic, local features of a marked object, such as its shape, color, and texture. *Relational aspects* register relational properties between marked objects, such as their relative shape, relative color, and relative position.

As an example, Figure 2 lists the specifications for the indexical sensory-motor system used by a program (to be described later) that learns to solve a simple block manipulation task. The system has two markers, the *action frame* and the *attention frame.* The action frame is used for both perception and action, while the attention frame is used only for perception. Each marker has a set of local aspects associated with it; these are the color and shape of the marked object, the height of the stack the marked object be-

[1] Agre and Chapman do not distinguish markers as overt or perceptual in their systems. The two classes were introduced because the learning algorithm described below depends on distinguishing between actions that change the world and actions that simply change the perception of the world.

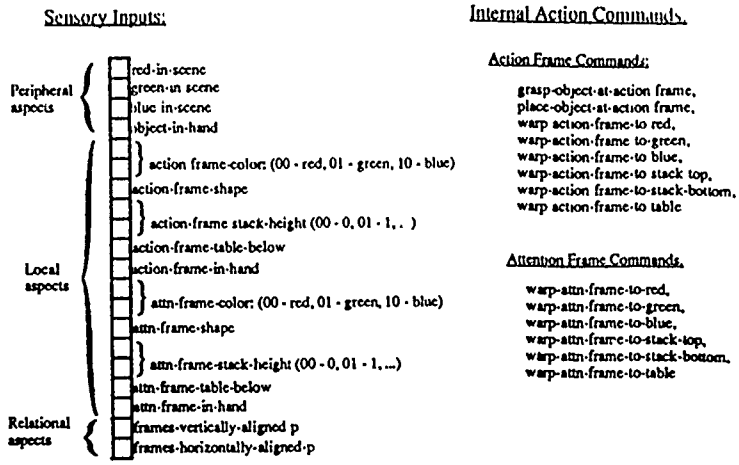


Figure 2: The specification for an indexical sensory-motor system containing two markers. The system has a 20 bit input vector, 8 overt actions, and 6 perceptual actions. The values registered in the input vector and the effects of internal action commands depend on the binding between markers in the sensory-motor system and objects in the external world.

longs to, whether or not the marked object is sitting on the table, and whether or not the marked object is held by the robot. The system has two relational aspects; one for recording vertical alignment between the two markers and one for recording horizontal alignment. Peripheral aspects include inputs for detecting the presence of red, green, and blue objects in the scene, and for detecting whether the hand is currently gripping an object.

The internal motor commands for the system are shown in Figure 2 on the right. The overt actions are made with respect to the action frame. The two primary overt actions are for grasping and placing objects. For grasping, *grasp-object-at-action-frame* causes the system to pick up the object marked by the action frame. The action works assuming the hand is empty and the marked object has a clear top.[2] Similarly for placing, *place-object-at-action-frame* causes the system to place an object its holding at the location of the action frame. This action also works assuming the hand is holding an object and the target object has a clear top. Other overt actions include actions for moving the action frame. Although these may appear to be perceptual actions, they are overt actions in the strictest sense since they affect the robots ability to perform other overt actions.

The attention frame is a perceptual marker and has a repertoire of perceptual actions that are used exclusively for gathering additional sensory information. As will be seen in Section 4, the attention frame plays an important role in allowing the agent to disambiguate world states.

All told the sensory-motor system has a 20 bit input vector: (4 bits of peripheral aspects, 14 bits of local aspects, 2 bits of relational aspects); and 14 actions (8 overt, 6 perceptual).

2.3 Reinforcement Learning

The task faced by the decision subsystem is a classic decision problem: given a description of the current state, a set of possible actions, and previous experience; choose the best next action. There are a number of approaches to this problem, but in this paper we are interested in decision systems that are based on reinforcement learning methods (RLM). In our experiments we have focussed on a representative reinforcement learning method known as *Q-learning* [Watkins, 1989], however our result regarding the interactions between RLMs and active perception apply to virtually all RLMs[Barto *et al.*, 1989].

A decision system based on Q-learning maintains two *inter*-dependent functions: an action-value function Q, and a policy function π. The action-value function, Q represents the system's estimate of the relative merit of making a given decision. That is, $Q(x, a)$ is the return the system expects to receive given that it executes action a in state x and follows its regular policy (π) thereafter. The policy function, π is the system's current estimate of the optimal decision policy. This function maps internal states ($x \in S_I$) into action commands ($a \in A_I$) and is defined in terms of the action-value function as follows:

$$\pi(x) = a \text{ such that } Q(x, a) = \max_{b \in A_I}(Q(x, b)) \quad (2)$$

That is, for a given state x, the policy function simply selects the action a, that (according to Q) maximizes the expected return.

Initially, the action-value function may be erroneous. However a procedure for incrementally improving Q can be obtained by recognizing that an accurate action-value function satisfies the following relation.

$$Q(x_t, a) = \mathrm{E}[R(X_{t+1}) + \gamma U(X_{t+1})] \quad (3)$$

where E() denotes the expectation, X_{t+1} is the random variable denoting the state at time $t + 1$, $R(x)$ is the reward the system receives upon entering state x, and $U(x)$ is called the *evaluation function* and is defined as

$$U(x) = max_{b \in A_I}(Q(x, b)) \quad (4)$$

Based on these equations, the following rule can be

[2] Although this isn't strictly necessary since, unlike traditional planning systems (e.g. STRIPS [Fikes and Nilsson, 1971]), the agent doesn't rely on any sort of internal model to predict the effects of actions but simply watches what happens in the world.

Decision cycle for 1-step Q-learning:

1) Generate a random number p, between 0.0 and 1.0,
2) if ($p < 0.9$)
   then $action \leftarrow \pi(x)$, where π is the policy function and x is the current state
   else $action \leftarrow R(A)$, where $R()$ is a random selection function,
3) Execute action, let x_{new} be the resulting state, and r the reward received
4) Compute the 1-step error:
   $error \leftarrow r + \gamma U(x_{new}) - Q(x, action)$
5) Update the action value of the selected decision:
   $Q(x, action) \leftarrow Q(x, action) + \alpha\, error$
6) Update the decision policy (for state x):
   $\pi(x) = a$ such that $Q(x,a) = max_{b \in A}[Q(x,b)]$
7) Update the evaluation function (for state x):
   $U(x) \leftarrow Q(x, \pi(x))$
8) Update the current state: $x \leftarrow x_{new}$
9) Goto 1

Figure 3: The steps in the decision cycle of a decision system based on 1-step Q-learning.

used for updating Q:[3]

$$Q_{t+1}(x_t, a) = Q_t(x_t, a) + \alpha(r_{t+1} + \gamma U_t(x_{t+1}) - Q_t(x_t, a)) \quad (5)$$

where Q_t is the action-value function at time t, r_{t+1} is the reward receives at time $t+1$, and α is a constant that affects the learning rate. This updating rule is known as the *1-step Q-learning rule*, and the term $r_t + \gamma U_t(x_{t+1})$ is called the *corrected 1-step truncated return*. Watkins has shown that under the appropriate conditions, a decision system using the 1-step Q-learning rule is guaranteed eventually to learn the optimal decision policy [Watkins, 1989].

Figure 3 outlines the control procedure for one simple decision subsystem implementing 1-step Q-learning. The first step in the procedure is to select the next action. 90% of the time the system selects the action specified by its control policy $\pi(x)$; the remaining 10% of the time it chooses an action at random. The action is then executed and the subsequent state and reward are noted. Once the effects of the action are known, the error in the action-value function for the current decision can be computed and used to update Q. Finally, $\pi(x)$ and $U(x)$ are updated to reflect changes in Q. The reason the decision system doesn't always select the action specified by its policy is that the action-value of a decision is only updated when that decision is executed. Occasionally selecting a random action insures that each decision will be evaluated periodically. In our experiments we implemented a similar but slightly more complex learning procedure that uses a weighted sum of n-step error terms and is based on Sutton's TD methods. (See [Sutton, 1988, Watkins, 1989] for details.)

3 Perceptual Aliasing

The straightforward integration of indexical representations and RLM decision systems lead to undesirable interactions that prevent the decision system from learning an optimal control strategy. These interactions arise because the mapping between world states and internal states is many to many. That is, a state $s_e \in S_E$ in the world, depending on the configuration of the sensory system, may map to several internal states; and conversely, a single internal state, $s_i \in S_I$, can represent several world states. We call this overlapping between world and internal state spaces *perceptual aliasing* and say an internal state is *perceptually ambiguous* if it can represent multiple world states. Figure 4 illustrates perceptual aliasing in a simple block world domain in which we adopt the sensory-motor system defined in Figure 2. Figure 4a shows two different world states (top) that generate the same internal representation simply because the markers are focussed on the part the states that are similar. In the figure the (+) represents the action-frame marker and the (*) represents the attention-frame marker. Similarly, Figure 4b shows that a single world state has multiple internal representations, each corresponding to a different placement of the attention-frame marker.

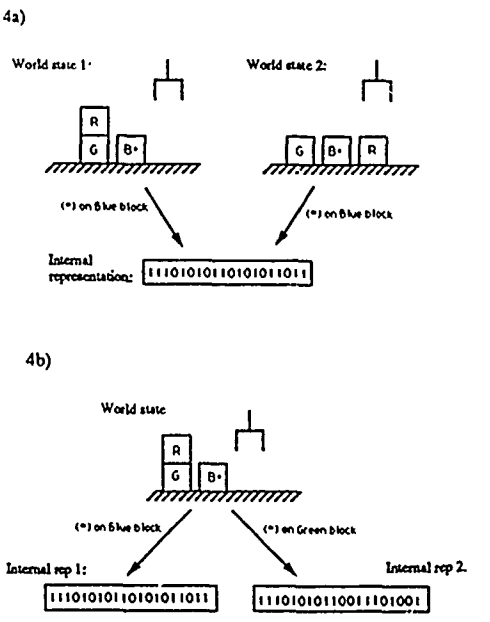


Figure 4: Generally the mapping between external world states and the agent's internal representation is many to many. a) shows how two different external states can generate the same internal representations and b) shows how one external state may have more than one internal representation.

Perceptual aliasing has a devastating impact on the decision system's ability to learn an adequate control policy because it causes the system to confound world states that it must distinguish in order to solve the task. The easiest way to illustrate the problem is to consider tasks in which the decision system receives a fixed reward upon achieving a particular goal

[3] See [Barto *et al.*, 1989] or [Watkins, 1989] for a detailed derivation.

state $g \in S_E$, and otherwise receives no reward. In this case, it can be shown that the state evaluations $U(s)$, and the action-values, $Q(s,a)$, of the steps in the optimal policy monotonically increase as the system approaches the goal state. Essentially, the optimal policy corresponds to a gradient ascent of the evaluation function. The upper graph in Figure 5 illustrates this result schematically for the optimal sequence $s_1, s_2, s_3, s_4, s_5, g$, for getting from s_1 to g. Each node in the graph represents a world state and each arc represents a possible decision. Solid arcs represent optimal decisions and dotted arcs represent non-optimal decisions. Shown below each state is its corresponding evaluation, $U(s)$, and below each decision is its corresponding action-value, $Q(s,a)$.

If the decision system has direct access to the world states, experience has shown that reinforcement learning methods can successfully learn an adequate if not optimal decision policy[4]. Unfortunately, the decision system does not have access to the state of the external world, but necessarily accesses the world via a relativized representation generated by the sensory system. This fact has been widely ignored in previous work on reinforcement learning. Existing systems ignore the issue of perception completely or assume a one to one mapping between the external state and the internal representation. Such simplifications are natural in toy domains but become painfully inadequate when scaling to larger problems. If the mapping between the external world and the internal prepresentation allows for perceptual aliasing, then the optimal policy may become unstable with respect to the learning algorithm. That is, because of perceptual aliasing, the learning algorithm will actually prevent the system from settling on the optimal policy.

The lower graph in Figure 5b shows how this instability arises. Consider the mapping between the world states and the internal states for the chunk of the state spaces shown. The mapping is one to one for all states except external states s_2 and s_5, which map to internal state s^i_a. If we fix the decision policy so that the system follows the optimal policy 90% of the time and chooses a random action 10% of the time and allow the system to estimate its evaluation function, U and action-value function, Q, by running the system for many trials, we find the following. First, since the state evaluation and action-value functions represent expected returns, for s^i_a they take on values somewhere between the corresponding values for s_2 and s_5. That is, $U(s_2) \leq U(s^i_a) \leq U(s_5)$ and $Q(s_2, a^i_r) \leq Q(s^i_a, a_r) \leq Q(s_5, a_r)$, where a_r is the optimal actions associated with s^i_a. Second, the state evaluations and action values do not monotonically

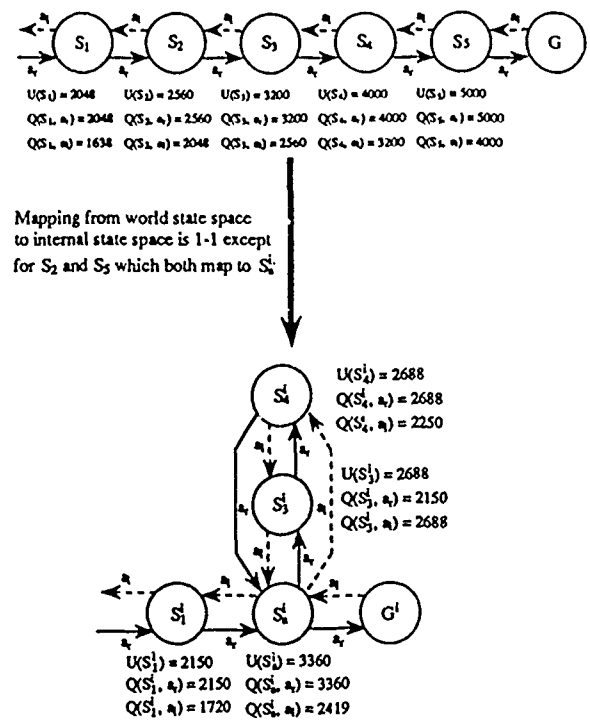


Figure 5: An example of the effect of perceptual aliasing on the evaluation function. The top graph shows a state transition diagram for a simple problem. In this case, the system receives reward only upon entering state G and the evaluation function monotonically increases as the distance from the goal decreases. The bottom graph shows the internal state space for the problem when the sensory system confounds states s_2 and s_5. In this case, the evaluation function is no longer monotonic and the optimal decision policy is unstable.

increase as the system approaches the goal. Instead a local maximum in the evaluation function arises at s^i_a. We call this maximum an *aberrational maximum* since it doesn't reflect the true utility of the underlying decision problem. If we relax our hold on the decision policy and allow the decision system to adapt, we find the optimal policy is unstable! Not only can't the system find the optimal policy it actually moves away from it. In general, the system will oscillate among policies, never finding a stable one.

4 Dealing with Perceptual Aliasing

The main result in this paper is a decision system based on reinforcement learning that can cope with perceptual ambiguity. The new algorithm leads to a control architecture that not only learns the correct overt actions needed to solve a problem, but also learns to focus its attention on the objects in the world that are relevant to the task. The design is based on three observations. First, in active perception a world state can be represented by multiple internal states,

[4] For 1-step Q-learning, Watkins has shown that appropriately designed decision systems can be guaranteed to converge on the optimal policy. Proofs for other, more general algorithms do not currently exist but many experimental systems have been built that learn optimal (or near optimal) policies [Anderson, 1989, Barto *et al.*, 1983, Clocksin and Moore, 1988, Watkins, 1989, Whitehead, 1989].

one of which is usually unambiguous. That is, in any given state, if the system looks around enough it will eventually attend to those objects that are relevant to the task and the internal state associated with that sensory configuration is unambiguous. Our algorithm depends on the existence of one such unambiguous internal state for each world state. Second, perceptually ambiguous states disrupt the decision system's ability to learn by promising erroneously large expected returns. If we can detect perceptually ambiguous states and actively lower their return estimates, we can minimize their negative affects. Third, if the world is deterministic, then perceptually ambiguous states will periodically overestimate their true evaluations, whereas the incidence of overestimation in unambiguous states diminishes with time. Therefore, perceptually ambiguous states can be detected by monitoring the sign of the estimation error in the updating rule, Equation 5.

The new algorithm is outlined in Figure 6. The system recognizes two classes of action; overt actions and perceptual actions. Overt actions change the state of the external world whereas perceptual actions change the mapping between world and the internal states [5]. The main decision cycle is the overt cycle which concerns itself with choosing overt actions in an attempt to maximize return. Embedded within the overt cycle is a perceptual cycle. After each overt action, the system executes a series of perceptual actions (the perceptual cycle) in an attempt to assess the true state of the external world. The objective of the perceptual cycle is to find an internal state that unambiguously represents the current world state. Upon completion, the perceptual cycle returns a list of the internal states encountered during the perceptual cycle, S_t. Each state corresponds to a different view (representation) of the current external world. The utility of the world state at time t, U_t, is estimated as the maximum utility of the individual internal states, $max_{s \in S_t}(U(s))$. As will be described below, our algorithm for adjusting the utility estimates of internal states severely lowers the utilities of perceptually ambiguous states. Consequently, utility estimates for world states tend to be based on actual utilities of unambiguous states and not biased by aberrational maxima associated with perceptually ambiguous states. Once U_t has been estimated, the Q estimates for the previous overt action are updated. The overt cycle then continues by selecting an overt action to execute. Some high fraction of the time (90%) the system chooses the action consistent with its policy; the rest of the time (10%) it chooses an action at random. When following policy, the action is chosen by searching among active internal states, S_t, for the decision with the largest action-value. Once an overt action is chosen, it is executed and the overt cycle begins anew.

[5] As a side effect, overt actions may also change the perceptual configuration, but perceptual actions are not allowed to affect the world state. In the indexical sensory-motor system described in Figure 2, the action-frame has only overt actions, and the attention-frame has only perceptual actions.

Overt Cycle:

1) Execute Perceptual cycle , generating S_t a set of internal representations for the current world state.
2) Estimate the utility of the current world state, $U_t \leftarrow max_{s \in S_t}[U(s)]$
3) Execute Update-Overt-Q-Estimates based on U_t, r_t, $o\text{-}act_{t-1}$, and $lion_{t-1}$, where r_t is the reward received at time t, $o\text{-}act_{t-1}$ is the last overt action executed, and $lion_{t-1}$ is the internal state selected to represent the previous world state (see below)
4) Decide if next action will follow policy or will be random.
   Follow policy 90% of the time.
   If following policy then $o\text{-}opt \leftarrow T$
   else $o\text{-}opt \leftarrow F$
5) Select next overt action:
   If $o\text{-}opt = T$ then
      i) $lion \leftarrow l \in S_t$ such that $U(l) = max_{s \in S_t}[U(s)]$
      ii) $o\text{-}act \leftarrow oc \in A_O$ such that $Q(lion, o) = max_{b \in A_O}[Q(lion, b)]$
   else
      i) $o\text{-}act \leftarrow R(A_O)$;;; choose action randomly
      ii) $lion \leftarrow l \in A_O$ such that $Q(l, o\text{-}act) = max_{s \in S_t}[Q(s, o\text{-}act)]$
6) Execute $o\text{-}act$ to obtain a new world state and goto 1.

Update-Overt-Q-Estimates:

1) Estimate the error in the lion's Q-value:
   $error \leftarrow (r_t + \gamma U_t) - Q(lion_{t-1}, o\text{-}act_{t-1})$
2) If ($error < 0$) then
   the lion is suspected of being ambiguous so suppress it.
   $Q(lion_{t-1}, o\text{-}act_{t-1}) \leftarrow 0.0$
   otherwise update it using the standard 1-step Q-learning rule:
   $Q(lion_{t-1}, o\text{-}act_{t-1}) = Q(lion_{t-1}, o\text{-}act_{t-1}) + \alpha\, error$
   also update non-lion internal representations
   for each $s \in S_{t-1}$ and $s \neq lion_{t-1}$ do
   $Q(s, o\text{-}act_{t-1}) = Q(s, o\text{-}act_{t-1}) + \alpha'\, error$;;; where $\alpha' < \alpha$

Perceptual Cycle:

1) Initialize S_t: $S_t \leftarrow \{s_c\}$, where s_c is the current internal state.
2) Do n times: (in our implementations $n = 4$)
   i) decide whether or not to follow policy
      follow policy 90% of the time
   ii) select next perceptual action
      If following policy then
      $p\text{-}act \leftarrow a$ such that $Q(s_c, a) = max_{b \in A_P}[Q(s_c, b)]$
      otherwise $p\text{-}act \leftarrow R(A_P)$
   iii) execute $p\text{-}act$ to obtain a new internal state s'
   iv) update Q estimate for the $p\text{-}act$
      $Q(s_c, p\text{-}act) = Q(s_c, p\text{-}act) + \alpha\,(U(s') - Q(s_c, p\text{-}act))$
   v) add s' to S_t: $S_t \leftarrow S_t \cup \{s'\}$
   vi) update s_c: $s_c \leftarrow s'$
3) Return S_t

Figure 6: An outline of the steps executed by the new decision system designed specifically to overcome the difficulties caused by perceptual aliasing.

The standard Q-learning algorithm defines the action-value of a decision as the return the system expects to receive given that system makes that decision and follows its policy thereafter, as characterized by Equation 3. However, for perceptually ambiguous states this definition leads to artificially high action-values (aberrational maxima). We have developed a modified learning algorithm that is based on Q-learning but incorporates a competitive component. This component tends to suppress the action values of perceptually ambiguous states while allowing action-values for unambiguous states to take on their nominal values. The result is that utility estimates for world states are more accurate since they are based on predictions from unambiguous internal states and not on perceptually ambiguous states.

Our modified learning algorithm is based on identifying one internal state, among S_t, that takes the lion's share of the responsibility (credit or blame) for the outcome of the current decision. We identify this

state as the *lion*. If the system is following its policy then the lion is defined as: $lion = s_l$ such that $Q(s_l, a_l) = \max_{s \in S_t, a \in A_O}(Q(s, a))$. When the system chooses a random action, a_{random}, the lion is defined as: $lion = s_l$ such that $Q(s_l, a_{random}) = \max_{s \in S_t}(Q(s, a_{random}))$. The idea underlying the use of a lion is that the lion state should be an internal state that unambiguously represents the current world state, and once such a state is found it is used to direct all actions associated with the world state it represents.

Perceptually ambiguous lions are detected and suppressed as follows. If the action-value associated with the lion, $Q(s_l, a_t)$, is greater than the estimated return obtained after one step, $r_t + \gamma U(s_{t+1})$, then the lion is suspected of being ambiguous and the action-value associated with it is suppressed (e.g reset to 0.0). Actively reducing the action-values of lions that are suspected of being ambiguous gives other (possibly unambiguous) internal states an opportunity to become lions. If the lion does not over-estimate the return it is updated using the standard 1-step Q-learning rule. To prevent ambiguous states from climbing back into contention, the estimates for non-lion states are updated at a much lower learning rate and only in proportion to the error in the lion's estimate. The observation that allows this algorithm to work is that ambiguous states will eventually (one time or another) overestimate action-values, consequently they will eventually be suppressed. On the other hand, it can be shown that an unambiguous lion is stable (i.e. will not over estimate its action-value) if every state between the lion's state and the goal also has an unambiguous lion. Thus, ambiguous states are unstable with respect to lionhood, while unambiguous states eventually become stable. The steps for updating action-values are shown in Figure 6 under the **Update-Overt-Q-Estimates** heading.

The steps in the perceptual cycle are sketched in Figure 6 under the **Perceptual Cycle** heading. The objective of the perceptual cycle is to accumulate a set of internal representations of the external world, one of which is unambiguous. This is achieved by executing a series of perceptual actions. In our current implementation, each perceptual cycle executes a fixed number ($n = 4$) of perceptual actions. This has proven adequate for our experiments, however it is easy to imagine variable length perceptual cycles, in which the cycle terminates as soon as an unambiguous internal state is found or increases when ambiguous states are encountered.[6] The algorithm for selecting actions within the perceptual cycle is similar to the algorithm for choosing overt actions in the overt cycle. The vast majority of the time, (90%), the system follows its policy and a small fraction of the time, (10%), a perceptual action is selected at random. When following policy, the action selected is the perceptual action a_p such that $Q(s, a_p) = \max_{b \in A_P}(Q(s, b))$ where s is the system's current internal state. That is, the policy calls for perceptual actions that lead to internal states with maximal expected returns.

[6] Actually, it may be possible to eliminate the distinction between the overt cycle and the perceptual cycle and integrate them into a single cycle, in which the action (either overt or perceptual) with the highest utility is chosen. We are currently experimenting with such an algorithm.

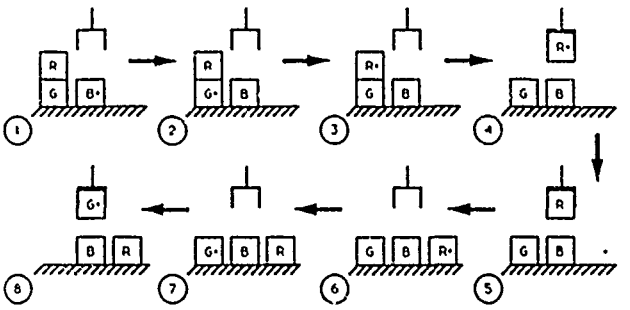


Figure 7: A sequence of world states in a typical solution path for the block manipulation task. Depending upon the placement of the attention frame, states 1, 3, 4, 5, and 6 may be represented ambiguously.

The rules for updating action-values for perceptual actions are those for standard 1-step Q-learning and are shown in Figure 6 within the **Perceptual Cycle** procedure. These updating rules lead to action-values that estimate the average utility of the states that result from executing a perceptual action. Since unambiguous states tend to have higher utilities than ambiguous states (which are suppressed), the effect is to choose perceptual actions that lead to unambiguous internal states.

5 An Example

To test our ideas we have implemented a system that learns a very simple block manipulation task. The task goes as follows The agent is presented with a pile of blocks on a conveyor belt. The agent can manipulate the pile by picking and placing blocks. When the agent arranges the blocks in certain *goal* configurations, it receives a fixed reward of 5000 units. Otherwise it receives no reward. When the agent solves the puzzle the pile immediately disappears and a new pile comes down the belt. If the agent fails to solve the puzzle after 75 steps, the pile falls off the end of the conveyor and a new pile appears at the front. A pile can have any number of blocks in it and can be arranged in arbitrary stacks. A block can be any one of three colors: red, green, or blue. The robot's sensory-motor system is the indexical system described in Figure 2. Also, we make the standard assumptions that a block can only be picked up if its top is clear and that a block can only be placed at locations with clear tops.

The particular task we studied rewards the agent whenever it picks up a green block. That is, goal configurations consists of of those states in which the

robot is holding a green block. We chose to study this task because it is very simple but adequate to demonstrate the difficulties cause by perceptual ambiguity. These problems can be seen in Figure 7 which shows the sequence of world states the agent traverses in solving one instance of a problem whose initial state consists a red block on a green block and a blue block on the table. Depending on the placement of the attention frame, world states 1,3,4,5, and 6 may have ambiguous internal representations. If the attention frame is fixed on the green block, then the states are unambiguously represented. However, if the attention frame is fixed on the blue block then the internal representation of the states is ambiguous. For example, in state 6, if the attention frame is fixed on the blue block then this state cannot be distinguished from other world states that are identical except with additional blocks above the green block. The system overcomes ambiguities by learning to direct its attention frame to the green block, which provides sufficient extra information (the height of the green stack) needed to disambiguate the situation.

Experiments were performed to obtain quantitative data on the new decision system's performance. Each experiment consisted of presenting the robot with a sequence of 500 piles of 4 randomly configured blocks, with each pile containing one green block. The robot's performance is illustrated in Figure 8. The figure shows the number of steps required to solve the problem (averaged over 20 runs of 500 piles) as a function of the number of trials seen. Initially the number of steps required is high, near the maximum of 75, since the robot thrashes around randomly searching for reinforcement. However, as the robot begins to solve a few problems its experience begins to accumulate and it develops a general strategy for obtaining reward. By the end of the experiment, the time required to solve the problem is close to optimal. The system's performance doesn't converge to optimal since 10% of the time it chooses a random action. This anomaly reflects a simplification in our decision algorithm that can be eliminated by incorporating a slightly more complex procedure for controlling exploration [Barto *et al.*, 1989].

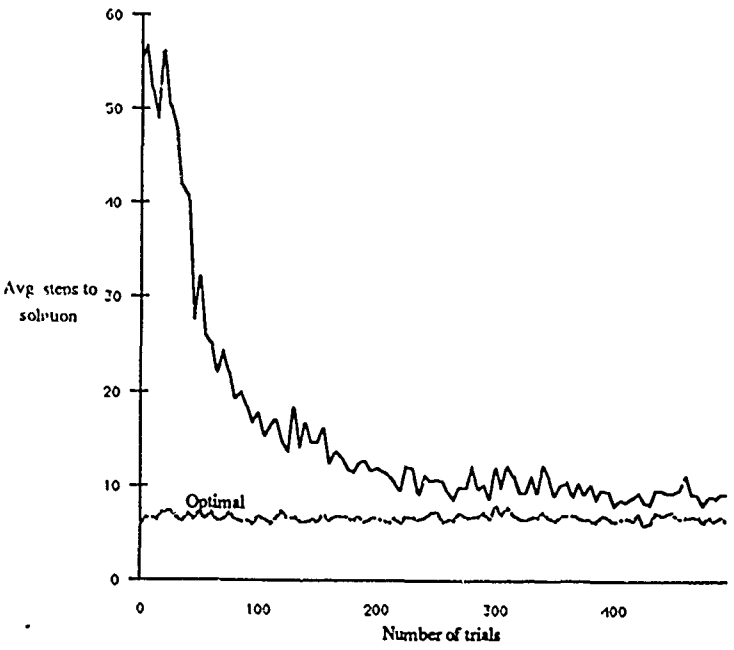


Figure 8: A plot of the average number of steps required to solve the block manipulation task as a function of the trails seen by the agent.

6 Conclusions

In this paper, we have considered the interactions that arise in adaptive control architectures that integrate active sensory-motor systems (specifically indexical representations) with decision systems based on reinforcement learning. We found the integration nontrivial since active sensory-motor systems naturally lead to internal states that are perceptually ambiguous. Perceptual ambiguity wreaks havoc on the decision system's ability to learn since it introduces aberrational maxima in the evaluation function and destabilizes the optimal policy. A solution to this problem was proposed that is based on actively detecting and suppressing ambiguous states. The result is a system that learns to focus its attention on the relevant aspects of the domain as well as control its overt behavior. The new algorithm was demonstrated in a system that learns a simple block manipulation task that is beyond the scope of previous reinforcement learning systems. Although our systems are still very primitive, we find the results encouraging and are hopeful that continued effort will yield systems capable of more sophisticated behavior.

6.0.1 Acknowledgments

We are thankful Josh Tenenberg for his insightful discussions throughout the development of the paper. We also thank Chuck Anderson, Chris Brown, and Rich Sutton for commenting on an earlier draft.

6.0.2 References

[Agre and Chapman, 1987] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *AAAI*, pages 268–272, 1987.

[Agre, 1988] Philip E. Agre. *The Dynamic Structure of Everyday Life.* PhD thesis, MIT Artificial Intelligence Lab., 1988. (Tech Report No. 1085).

[Anderson, 1989] Charles W. Anderson. Towers of hanoi with connectionist networks: learning new features. In *Proceedings of the Sixth International Conf. on Machine Learning*, pages 345–350, Ithaca, NY, 1989. Morgan Kaufmann.

[Barto and Sutton, 1981] Andrew B. Barto and Richard S. Sutton. Landmark learning. An illustration of associative search. *Biological Cybernetics*, 42:1–8, 1981.

[Barto *et al.*, 1983] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuron-like elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.

[Barto *et al.*, 1989] Andrew B. Barto, Richard S. Sutton, and Chris Watkins. Sequential decision problems and neural networks. In *Proceedings of 1989 Conf. on Neural Information Processing*, 1989.

[Blythe and Mitchell, 1989] Jim Blythe and Tom M. Mitchell. On becoming reactive. In *Proceedings of the Sixth Intl. Conf. on Machine Learning*, pages 255–259, 1989.

[Booker, 1982] Lashon B. Booker. *Intelligent behavior as an adaptation to the task environment.* PhD thesis, University of Michigan, 1982.

[Brooks, 1986] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, pages 14–22, April 1986.

[Chapman, 1989] David Chapman. Penguins can make cake. *AI Magazine*, 10(4):45–50, 1989.

[Clocksin and Moore, 1988] W. F. Clocksin and A. W. Moore. Some experiments in adaptive state-space robotics. Technical report, Computer Laboratory, University of Cambridge, October 1988.

[Drummond, 1989] Mark Drummond. Situated control rules. In *Proceedings of the Rochester Planning Workshop*, 1989. (Also Univ. of Rochester Techical Report No. 284).

[Fikes and Nilsson, 1971] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Fikes *et al.*, 1972] Richard E. Fikes, Paul E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artifical Intelligence*, 3(4):251–288, 1972. Also in Readings in Artificial Intelligence, 1980.

[Firby, 1987] R. James Firby. An investegation into reactive planning in complex domains. In *AAAI*, pages 202–206, 1987.

[Georgeff and Lansky, 1987] Michael P. Georgeff and Amy L. Lansky. Reactive reasoning and planning. In *AAAI*, pages 677–682, 1987.

[Holland, 1986] John H. Holland. Escaping brittleness: the possibilities of general purpose learning algortihms applied to parallel rule-based systems. In *Machine Learning: An Artificial Intelligence Approach. Volume II.* Morgan Kaufmann, San Mateo, CA, 1986.

[Kaelbling, 1989] Leslie P. Kaelbling. A formal framework for learning in embedded systems. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 350–353, 1989.

[Laird *et al.*, 1986] John E. Laird, Paul S. Rosenbloom, and Alan Newell. Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning*, 1(1):11–46, 1986.

[Miller *et al.*, 1990] W. T. Miller, R. S. Sutton, and P. J Werbos. *Neural Networks for Control.* MIT Press, Cambridge, MA, 1990.

[Nilsson, 1989] Nils J. Nilsson. Action networks. In *Proceedings of the Rochester Planning Workshop*, 1989. (Also Univ. of Rochester Techical Report No. 284).

[Schoppers, 1987] Marcel. J. Schoppers. Universal plans for reactive robots in unpredictable domains. In *Proceedings of IJCAI-87*, 1987.

[Schoppers, 1989] Marcel J. Schoppers. *Representation and Automatic Synthesis of Reaction Plans.* PhD thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 1989.

[Sutton, 1984] Richard S. Sutton. *Temporal Credit Assignment In Reinforcement Learning.* PhD thesis, University of Massachusetts at Amherst, 1984. (Also COINS Tech Report 84-02).

[Sutton, 1988] Richard S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[Sutton, 1990] Richard S. Sutton. First results with DYNA, an integrated architecture for learning, planning, and reacting. In *Proceedings of the AAAI Spring Symposium on Planning in Uncertain, Unpredictable, or Changing Environments*, 1990.

[Watkins, 1989] Chris Watkins. *Learning from delayed rewards.* PhD thesis, Cambridge University, 1989.

[Whitehead and Ballard, 1989a] Steven D. Whitehead and Dana H. Ballard. Reactive behavior, learning, and anticipation. In *Proceedings of the NASA Conference on Space Telerobotics*, Pasadena, CA, 1989.

[Whitehead and Ballard, 1989b] Steven D. Whitehead and Dana H. Ballard. A role for anticipation in reactive systems that learn. In *Proceedings of the Sixth International Conference on Machine Learning*, Ithaca, NY, 1989. Morgan Kaufmann.

[Whitehead, 1989] Steven D. Whitehead. Scaling in reinforcement learning. Technical Report TR 304, Computer Science Dept., University of Rochester, 1989.

[Wilson, 1987a] Stewart W. Wilson. Classifier systems and the animate problem. *Machine Learning*, 2:199–228, 1987.

[Wilson, 1987b] Stewart W. Wilson. Hierarchical credit allocation in a classifier system. In *Proceedings of the Tenth IJCAI*, pages 217–220. Morgan Kaufmann, 1987.

LEARNING AND PLANNING

Learning Plans for Competitive Domains

Susan L. Epstein*
Department of Computer Science
Hunter College of the City University of New York
New York NY 10021

Abstract

Many machine learning systems learn from their problem solving experience in single-agent domains. This paper discusses an algorithm to learn complex, relevant, cost effective plans for a broad class of competitive, multi-agent domains. Such a plan, called a fork, is extracted from the explanation of a single failure, and represents a set of mutually overlapping simple plans to achieve a goal. Each plan is non-linear, provides alternatives for contingencies, is applicable both offensively and defensively, and has a clear upper bound for its execution time. This paper describes the representation and implementation of forks in HOYLE, a system to learn to play any two-person, perfect information game well. Together with a weak theory for its general domain, HOYLE has used forks to learn to play a broad variety of games perfectly without extensive forward search in the game tree. In more challenging games, however, the selection of a relevant plan and its binding to the current game state is unacceptably costly. This paper details the significantly improved performance directly attributable to learning about appropriate forks, and the heuristics that guard against unacceptable degradation of performance as new knowledge is acquired.

1. Introduction

From their problem solving experiences, many machine learning programs extract and reformulate information that can improve their performance in single-agent domains. Many of these programs (e.g., Fikes, Hart, & Nilsson, 1972; Korf, 1985; Laird, Rosenbloom, & Newell, 1986) have learned macro-operators for planning. Others (e.g., Langley, 1985; Minton, 1988; Mitchell, Utgoff, & Banerji, 1983) have learned heuristic rules to improve their search performance in planning domains. Recent research in explanation-based learning (e.g., Minton, 1988; Mooney, 1988) has demonstrated how one problem-solving example can provide relevant, cost effective knowledge within a broad domain. In a domain with more than one agent, however, credit and blame assignment is more difficult, the next choice point for each agent is not absolutely predictable, and the absolute merit of goal states and paths to them is difficult to assess. There has been little work on learning useful plans for broad domains with more than one agent. Minton (1984) described an algorithm that learned recognition rules for plans in a two-agent domain. These rules, however, were limited to chess and slowed the system "dramatically."

This paper describes an algorithm to learn a class of extremely powerful plans in a domain where two or more adversarial agents perform some sequence of unretractable, possibly interfering actions in a race to achieve some goal. The algorithm has been implemented within HOYLE, a machine learning program for two-person, perfect information games (Epstein, 1989a). The plans are called forks; they are game-independent, partially ordered, and applicable both offensively and defensively. Each plan provides alternatives for contingencies and has a known upper bound on its execution time. Together with a weak theory for its general domain, HOYLE has previously used such preselected forks to learn to play a broad variety of games perfectly without extensive forward search in the game tree. In more challenging games, however, the selection of a relevant plan and its binding to the current game state is unacceptably costly. The algorithm discussed here enables HOYLE to learn a relevant plan as an explanation of a single losing experience, and to improve its performance significantly. Heuristics guard against unacceptable degradation of performance as new knowledge is acquired, and support the correct offensive and defensive application of these plans.

2. An Overview of HOYLE

HOYLE is a system that learns to play a broad class of two-person, perfect information games extremely well.

*This work was supported in part by NSF DCR-8514395 and PSC-CUNY 668287 and 666397.

A *two-person, perfect information game* is one in which all information about the game is disclosed and equally available to its two participants, e.g., there are no closed hands and no uncertain outcomes as with dice. Any such game can be represented as a finite, directed graph (the *game graph*) in which a node (*state*) both identifies the participant whose turn it is (the *mover*) and describes a possible arrangement of the *material* (the board and playing pieces). For the games considered here, there are exactly two participants: *Player* (the one who moves first) and *Opponent*. Player and Opponent are the *roles* in a two-person game. During a *contest* (a path in a game graph that represents one complete experience of the game), Player tries to arrive at a terminal node categorized as a win and Opponent tries to arrive at a terminal node categorized as a loss.

Since from any state there are usually many legal moves, the challenge in play is for the mover to select the best move, i.e., one that will maximize the mover's opportunity to arrive at a desired result, while minimizing the non-mover's chances to arrive at her own desired result. *True expertise* at a game is perfectly backed up knowledge from its entire game graph. Most interesting games have extremely large game graphs; for them, true expertise through exhaustive search of the game graph is computationally intractable. Instead, computer game playing programs usually approximate true expertise by heuristic search in the game graph. From a given node, such a program estimates the unseen finishes of deeper intermediate nodes and backs up this approximate knowledge through the game graph to the given node to select the best possible legal move. Human experts can foil such programs by executing plans whose threat lies at a deeper level than the programs search.

HOYLE accepts as input a representation of some game: its material, interface, and rules. The interface provides the graphics for the game and interactive communication with another participant through the keyboard. The rules describe the legal moves, when a contest at the game terminates, and how to determine the outcome of that contest. From this input HOYLE is able to play any such game correctly, i.e., according to the rules; HOYLE's task is to learn to play it perfectly. Without perfect knowledge, the performance of a game-playing program must be evaluated empirically, by playing contests. HOYLE is evaluated against two criteria: how well it learns to play in a *tournament* (a sequence of contests where the participants alternate roles) against a human expert and how quickly it achieves that skill. Measures of skill include ability to win, ability to draw, and the duration of contests played, measured in average number of moves.

HOYLE is a limitedly rational program that learns from its experience. Compared to traditional one-game programs (e.g., Anantharaman, Campbell, & Hsu, 1988; Berliner, 1980; Berliner & Ebeling, 1989; Lee & Mahajan, 1988; Rosenbloom, 1982; Samuel, 1967; Schaeffer, 1988), HOYLE does relatively little forward search into the game tree, has a limited memory, and has no heuristics for symmetry. Instead, HOYLE makes decisions based on a variety of advice. HOYLE's thesis is that, in learning to play any new game, a person actually has relevant prior knowledge about it, a weak theory for the general domain of two-person, perfect information games. This weak theory encapsulates the person's experience from contests at other games, and enables her to learn to play any such game well. HOYLE's weak theory includes a set of valid, but deliberately narrow, viewpoints (HOYLE's *Advisors*) that are modified by HOYLE's experience, i.e., learn. When it is HOYLE's turn to move, each Advisor generates one or more comments about the current state. To select a move, HOYLE weighs the comments carefully against each other (Epstein, 1989b) according to the game, the contest, and the nature of the participants. Among HOYLE's Advisors is Pitchfork, a procedure that exploits certain very powerful plans, called forks. The way this Advisor learns the particular forks effective in a given game is the subject of this paper.

3. An Example of a Fork in Execution

Assume a domain where a finite set of actions is available, two or more agents alternately take a single action, and no action may be taken by more than one agent. A *simple plan* in such a domain is an unordered set of unretractable actions that achieve a goal. Informally, a fork is two or more overlapping simple plans belonging to the same agent, directed toward different goals, and having one or more common actions. (Although the implementation described here is for games and two agents, the algorithm is limited only by the definitions of simple plan and fork.) Figure 1 shows an example of a fork used in tic-tac-toe where Player's most recent move (action), in position 3, threatens a win

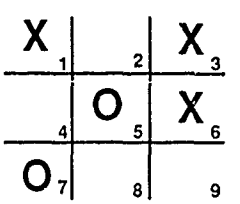


Figure 1: A Fork

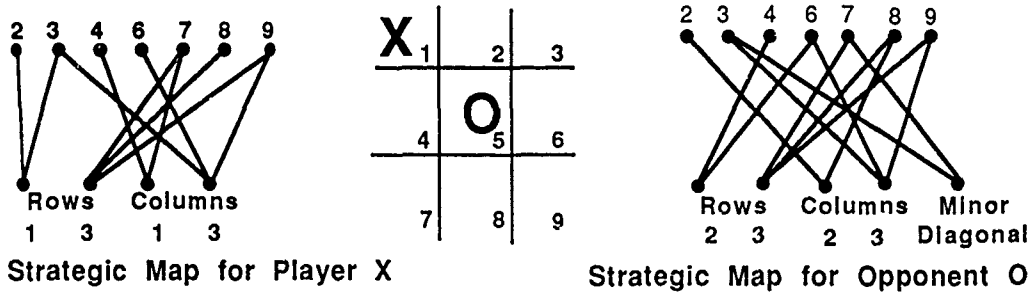


Figure 2: A Game State and Its Strategic Maps

in both the first row and the third column (the goals). Opponent is defenseless before this onslaught.

Banerji (1980) devised a graph representation for the mover's situation in a game focused upon certain patterns called "kernels." An early implementation of Banerji's ideas (Koffman, 1968; King, 1970) used a few simple kernels to suggest moves in several games, like tic-tac-toe. HOYLE has continued this research: it extends Banerji's original terminology, defines and explains the significance of a fork's depth, explores the generation and application of forks, and integrates forks with other techniques to support both *offensive* (goal-forwarding) and *defensive* (goal-inhibiting) play (Epstein, 1990).

4. Definition and Representation of Forks

In a task with only unretractable actions (actions from states that do not lie on a cycle in the search space), a strategic map for any state can be constructed for each agent. A *strategic map* for a state and agent is a labeled, undirected bipartite graph in which one set of nodes (the *top vertices*) is labeled with the actions available to that agent from that state, the second set (the *bottom vertices*) is labeled with the agent's simple plans to achieve her goal and an edge joins any action available to an agent with each simple plan it forwards. Thus the strategic map for an agent at a state represents all the goal-forwarding options open to her. A strategic map may be denoted as <T,B,E>, where T is a set of top vertices, B a set of bottom vertices, and E a set of undirected edges from vertices in T to vertices in B. In Figure 2, for example, positions 2, 3, 4, 6, 7, 8, and 9 are available to both participants; rows 1 and 3 and columns 1 and 3 are simple plans for Player; and rows 2 and 3, columns 2 and 3, and the minor (from upper right to lower left) diagonal are simple plans for Opponent. Figure 2 shows the strategic maps for Player and Opponent for the given board position.

Certain connected subgraphs of a strategic map, induced by a subset of its bottom vertices, represent the overlapping of more than one of the agent's simple plans. These subgraphs are denoted here as F-dn, where F stands for "fork," d is its depth, to be defined shortly, and n provides a distinct label. For example, in a subgraph of the form F-31 in Figure 3(a), either action represented by a degree-two vertex at the top furthers both the simple plans represented by the bottom vertices adjacent to it. Such an action, on the left for example, transforms F-31 into Figure 3(b), two connected graphs, of the forms F-1 on the left and F-21 on the right. (Immediately before her most recent move into position 3 in Figure 1, Player's strategic map was isomorphic to F-21, where the top vertices represented 2, 3 and 9 and the bottom vertices represented Row 1 and Column 3.) Subsequently taking the action represented by the second degree-two position transforms Figure 3(b) into Figure 3(c), three copies of F-1. Thus there are two actions represented in F-31 that, taken in any order, forward among them three simple

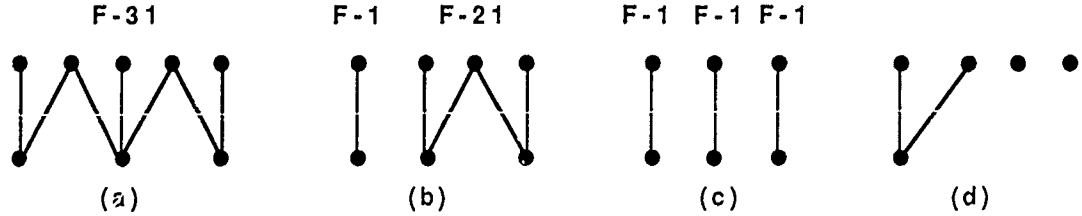


Figure 3: Applications of the Fork F-31

plans.

Formally, a *fork* is defined to be either F-1 = <{t},{b},{(t,b)}> with distinguished vertex (*key*) t, or an undirected bipartite graph containing at least one distinguished top vertex whose removal, along with its edges, disconnects the graph into two or more connected components, each of which is also a fork. The set of all forks is completely and correctly defined recursively as:

- F-1 is a fork.
- The following construction produces a fork: Copy any n forks. (They are the *parents* of the new fork under construction.) Add a new top vertex v. (This is the *join* of the new fork under construction.) Add an edge from v to at least one bottom vertex in each parent.
- Nothing else is a fork.

Observe that removal of the join and its associated edges transforms the graph into n connected graphs, its parents, each of which is itself a fork. F-21, for example, is the result of joining two copies of F-1. The removal of a key and its associated edges is called an *execution* of the fork.

The fork F-1 has depth 1, and for d >1, a *fork of depth d* is a fork such that removal of any one of its distinguished vertices produces at least one connected component that is itself a fork of depth d-1, and d is maximal. By this definition, for example, F-21 is a fork of depth two and F-31 is a fork of depth three. When some subset of the bottom vertices of an agent's strategic map induces a copy of a fork, the depth of that fork may be interpreted as an upper bound on the number of actions until the completion of one of those goals by the agent. For example, if F-31 is such an induced subgraph, the agent is certain to achieve one of her goals after three actions. (Reaching a goal state sooner is possible, of course, but against optimal defense, success will require three actions.) The keys and depth of a fork are non-trivial computations, however (Epstein, 1990). It is not the case, for example, that every fork built by combining two smaller forks takes longer to execute than its components. How quickly a fork can achieve its objective, quantified here as its depth, must be computed as the minimum of all fork depths arising in the map when the first agent takes any top vertex and then the other agent takes another.

In summary, a fork of depth d represents a game-independent set of mutually overlapping simple plans for a single agent to achieve a goal after at most d actions. The existence of at least one key in every fork imposes a partial ordering on the plan, and provides alternative plans: the parent forks that appear as vertices are removed. The power of a fork lies in the ability of one action to forward more than one plan.

5. The Application of Forks in Two-agent Domains

Because each agent in every state has a strategic map, forks are applicable as both offensive and defensive plans. In many search spaces, plans that guarantee the achievement of a goal state with d actions can be delayed, or even destroyed, by defensive interference from another agent. The best defense against any opposing fork is to take the action in one of its keys yourself, permanently eliminating many of the other agent's simple plans. For example, if one agent has a strategic map containing F-31 from Figure 3(a), the other agent can destroy the fork by taking the action in either of the degree-two keys (say, the right one), immediately reducing that strategic map to Figure 3(d). In what remains of the first agent's strategic map, the only remaining simple plan is readily prevented on the other agent's next turn. For an extended example of how this technique results in highly skilled play see (Epstein, 1990).

Strategic maps must be thoughtfully applied in the play of two-person, perfect information games. If the mover has more than one fork in her strategic map, the best offensive move (action) is always a key in the shallowest fork, that is, pursuit of her shortest certain plan to a win. (If F-1 is detected, this reduces to the common sense theory "If you see a winning move, take it.") If one participant's strategic map contains a fork F of depth d, a win in d turns is guaranteed, even when met by optimal defense, *but only when the second participant forwards no offensive plans of her own and it is the first participant's turn.* If the second pursues a shallower fork of her own, the first will be forced repeatedly to defend against it, delaying or perhaps even losing the ability to pursue F if the set of moves is small. If the second participant is the mover and plays a key in F, the first will have to search elsewhere for a fork.

The application of forks to two-person, perfect information games requires extensive computation, both before and during play. For a machine to exploit forks to their full advantage, it would have to identify correctly every fork, its depth, and its keys in both strategic maps. One way to recognize forks is to maintain a library of them, and search for subsets of bottom vertices in each strategic map that induce a subgraph isomorphic to an element of the library. Implementation of the recursive constructive definition of a fork to build such a library is non-trivial, however. The definition generates infinitely many forks at any given depth, can generate isomorphs of the same fork, and provides only an upper bound on the depth of the new fork. HOYLE's initial foray into

two-parent fork construction through depth four demonstrated the anticipated a combinatoric explosion. From the single fork F-1 at depth one, HOYLE built one fork at depth two, 5 distinct forks at depth three, and 520 distinct forks at depth four. Ideally, for any given contest state, HOYLE should construct its two strategic maps and then search for forks in them. Search for induced isomorphic subgraphs is notoriously expensive, however, well beyond the resource allocation of a limitedly rational system like HOYLE. Instead, a set of heuristics was developed for HOYLE to capitalize on a limited library of two-parent forks, both offensively and defensively.

HOYLE's heuristics exploit the strategic map in a variety of ways. HOYLE searches only for two-parent forks and only in connected components of a strategic map. The program divides its allocated resources approximately equally between offensive search of its own strategic map and defensive search of the other participant's strategic map. Search heuristics seek the shallowest forks in both maps, one depth value at a time. Each fork is screened before search to confirm that the number of its top vertices does not exceed the current number of legal moves in the state and that no bottom vertex is of degree larger than the number of positions required for a win. Finally, Advisors other than Pitchfork have access to the strategic maps it constructs; they recommend moves based on properties of the strategic maps and subgraphs of them. Further details and a complete algorithm appear in (Epstein, 1990).

For the simplest games in HOYLE's testbed, this approach, in combination with the other Advisors and the control structure, was able to learn to play with true expertise. For more difficult games, particularly those

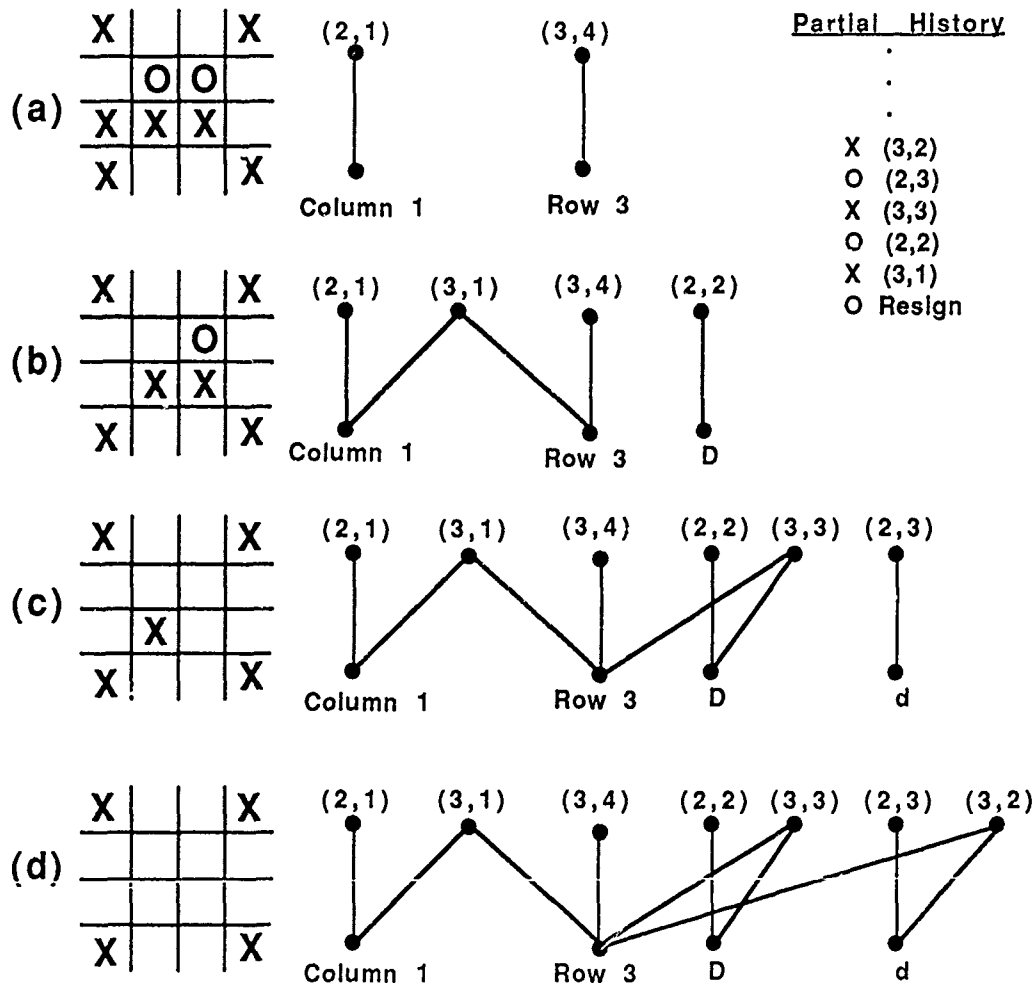


Figure 4: Learning a Fork

with larger boards, exhaustive search for induced isomorphic subgraphs is far too slow.

6. An Algorithm that Learns Forks

When it is HOYLE's turn to move, its Advisor Panic looks to see if the non-mover has a move that would win the game (called a *threat*). If Panic finds any threats, it looks for moves that would make all of them impossible on the non-mover's next turn. In the event that Panic detects a threat it cannot block, it correctly tells HOYLE to resign, because it is overwhelmed. This resignation is recorded in the history, the trace of the contest. In a positional game (one like tic-tac-toe where territory is occupied and cannot be released), a resignation by Panic is tantamount to an admission that the program has been forked. HOYLE takes a history of such an experience, and backs up two states at a time to explain its defeat. This procedure uses the constructive definition of fork in Section 4 to explain how the loss was due to the execution of a fork and precisely which fork it was. The learned fork, together with the heuristics HOYLE has for applying forks, is fully operational.

Consider, for example, the partial history of a Qubic contest HOYLE lost as shown in Figure 4. Qubic is a three-dimensional version of tic-tac-toe played on four parallel four-by-four grids, where a win is four in any straight line. Board positions in the plane shown are referenced here by matrix coefficients; the other planes, where the other four O's lie, are omitted. The major diagonal, from the upper left to the lower right, is denoted as D, and the minor diagonal as d. In Figure 4(a) it is HOYLE's turn to move as Opponent (O's), but the strategic map for Player (X's) indicates that there are threats at (2,1) and (3,4); taking Panic's advice, HOYLE resigns. Now HOYLE constructs an explanation of its loss, beginning with Player's final strategic map of two simple plans. The graph in Figure 4(a) is isomorphic to two copies of F-1. Two moves backward at a time, HOYLE modifies this graph until the history is exhausted (in which case there was no fork) or the graph is connected. To move from Figure 4(a) to 4(b), for example, HOYLE retracts Player's last move, (3,1), and HOYLE's response, (2,2), to Player's earlier threat on the minor diagonal. When HOYLE backs up, it adds to the graph as top vertices any moves that will add at least one edge from them to new simple plans of degree one or to preexisting simple plans. Thus the graph in Figure 4(b) is a copy of that in 4(a), with the addition of the vertices (3,1) and (2,2), the edge to the new simple plan D, and the edges to the preexisting simple plans for the first column and the third row. The graph in Figure 4(b) is isomorphic to a copy of F-1 and a copy of F-21. Similarly, the graph in Figure 4(c), isomorphic to a copy of F-31 and a copy of F-1, is before Player's move in (3,3) and HOYLE's (2,3) block of the threat on the minor diagonal. Finally, the graph in Figure 4(d) is the result of withdrawing Player's move in (3,2) and whatever move HOYLE had made immediately before it. (That move introduced no new edges to the graph.) At this point, reconstruction stops because the graph in Figure 4(d) is connected, i.e., HOYLE has learned a fork of depth 4.

Without its labels, the fork in Figure 4(d) is both a generalization of HOYLE's experience in a contest and a specialization of its recursive concept definition for forks. Empirically, this plan has been found quite useful both offensively and defensively in other Qubic contests. Because it is game-independent, it could be relevant to other games as well.

7. Measuring Learning in a Tournament

Learning in a tournament can be measured by changes in performance. When a game is well-understood mathematically, the cumulative number of wins and draws for true expertise can be plotted as a function of the number of contests played (the goal curve). In a HOYLE tournament, the cumulative number of wins and draws for the program is calculated as a function of the number of contests played (the performance curve). While the rate of change in the distance between a performance curve and the goal curve is decreasing, the program is learning to play better. Once its performance curve consistently parallels the goal curve, HOYLE may be said to have learned to play with true expertise. If a program loses repeatedly, but is able to prolong its contests as time passes, it is learning; a program that wins or draws repeatedly and is able to achieve that result more quickly as time passes is learning too.

Qubic is a challenging game. There are 76 possible winning configurations to guard against, and well over a billion possible states. There is an average of 32 possible legal moves from any state. It has been shown mathematically (Paul, 1978) that every contest between two participants with perfect knowledge should end in a tie. Forks of at least depth five are constructable in Qubic, well beyond the average human's notice, and outside Pitchfork's standard library as well.

HOYLE has played several tournaments of 20 Qubic contests each against an expert human, with a resource limit of three minutes per Advisor. In one tournament (#1) HOYLE played at random, making legal moves without any of its learning facilities, while the human pursued any simple plan. HOYLE lost every contest

very quickly, averaging 7.6 moves out of a possible 64, a lower bound on performance. In another tournament (#2), the human expert repeatedly played the depth four fork of Figure 4 against the full original implementation of HOYLE with a library of forks only through depth three. During #2, Pitchfork struggled to identify forks within its time constraints and regularly signaled that it was not allocated sufficient resources. By the point in the contest that Pitchfork, with its limited resources, detected an offensive fork, it was too late to defend against it, and HOYLE conceded, still losing every contest. The average contest in #2 was 19.4 moves, however; HOYLE offered much stronger competition than mere random play. The handicaps of limited rationality (both as a time limit and as a fork depth limit) and lack of knowledge of symmetry were too great for this version of HOYLE to make any visible progress at learning Qubic. Against a set of strong game players who were not Qubic experts, however, this version of HOYLE always won.

At this point HOYLE was revised as follows. Pitchfork was modified to learn forks, as described here, and instructed to consider first any recently-learned forks applicable to the current game. Pitchfork was allocated five minutes of computation time. (Each Advisor is otherwise allocated one minute and typically uses only a few seconds.) Pitchfork was told to recommend any good moves found, even those based on partial search of the strategic maps.

During initial forays at Qubic, this modified program learned several different forks of depth greater than three. At least one of these was more elaborate than the pattern the human expert had in mind. The extended time allocation slowed some of the early and non-forced moves, but still permitted real time play; no contest ran more than an hour.

The revised version of HOYLE, without this initial experience, played a tournament of 10 contests (#3). In the first contest of #3, HOYLE was defeated by the successful fork from Figure 4. After the contest, however, Pitchfork analyzed the history and learned the fork, placing it first on Pitchfork's search list. In subsequent contests, each time the human expert attempted that fork, HOYLE identified and blocked it successfully. Play was at an extremely high level throughout #3; contests now averaged 39.8 moves and competition was intense. In the fourth contest, for example, both participants had the fork from Figure 4 in their strategic maps (HOYLE's was a version using central spaces rather than the corners it had learned the fork on). Neither participant was able to execute the fork because of interference from the other, and the contest was a draw. In the seventh contest, the human expert in one state had two different isomorphs of the fork from Figure 4. HOYLE was able to block these, execute successfully a shallower fork of its own, and go on to win the contest. HOYLE never lost a contest in #3 after the first one, and it regularly caught the human expert off guard, winning six times.

HOYLE was also tested on a 3-by-3-by-3 version of tic-tac-toe, a somewhat simpler game that Player should always win. The original program learned to play perfectly after several contests. The fork-learning version learned to play with true expertise just as quickly, but it also learned the depth-four fork produced by the opening move in the post mortem for the first contest. (This fork is different from the one learned in Qubic.) In the remainder of the tournament, as Opponent HOYLE resigned after the correct opening; as Player HOYLE declared victory before it made a single move, but still went on to play perfectly and win each contest.

8. Results and Future Work

Previously, a few sets of mutually overlapping simple plans with very limited applicability were identified as naive offensive strategies in two-agent domains. HOYLE's game-independent, graph representation for forks includes these plans and provides a plan generator. Each fork implicitly contains both conjunctive and disjunctive descriptions. Proper application of the seven smallest forks has been shown empirically to produce high-quality solutions and to focus attention on strategic possibilities not otherwise likely to be found in some large search spaces.

For relatively easy games, simple heuristics to construct, recall, and search for these seven forks in a contest will support learning to play perfectly. For more difficult games, however, high match cost and utility (Minton, 1988) become issues. Empirical work with HOYLE thus far indicates that, perhaps by the topology of their boards and the nature of their rules, most games intrinsically have a limited number of applicable forks. Each larger fork relevant to a particular game is readily identified from an explanation of the trace of a defeat at that game. Searching first for these learned forks in subsequent contests of the same game suffices to simulate expert play.

The learned plans are generalizations potentially applicable to any game, with a well-defined metric (depth) that supports their error-free application both offensively and defensively without forward search into the game graph, and an implicit game-independent execution plan that provides for contingencies. Because heuristics are used in the implementation, performance

may be imperfect. In its current game testbed, however, HOYLE's ability to learn plans from observing its defeat at the hands of an expert rapidly enables it to learn to win. Finally, although the abstractions representing forks were designed for a particular class of games, they are clearly extendible to any multi-agent competitive planning domain.

Future research includes the extension of this work to contests where the winner deliberately deflects attention from a fork being executed, to games where moves are retractable, and to games where the goal is disabling playing pieces rather than achieving a specific configuration.

These results, while promising, raise an interesting issue: at the current time, HOYLE is not an aggressive player, merely an opportunistic one. On defense, HOYLE detects and blocks the forks it knows, within its heuristic constraints. On offense, however, HOYLE does not actively plan to construct a state in which it will have a fork; it only executes forks that happen to lie in a state. This opportunistic behavior is not aggressive play, or even aggressive planning. Current research includes *near forks*, aggressive plans that give rise to executable forks. Until HOYLE takes the construction of a fork, rather than its execution, as a goal, the program will continue to play excellent defense but only serendipitous offense.

References

Anantharaman, T., Campbell, M. and Hsu, F. 1988. Singular Extensions: Adding Selectivity to Brute Force Searching. In *Proceedings on Computer Game Playing*, 8-13. *AAAI 1988 Spring Symposium Series.*

Banerji, R. B. 1980. *Artificial Intelligence: A Theoretical Approach.* New York: North-Holland.

Berliner, H. J. 1980. Backgammon Computer Program Beats World Champion. *Artificial Intelligence* 14 (2): 205-220.

Berliner, H. and Ebeling, C. 1989. Pattern Knowledge and Search: The SUPREM Architecture. *Artificial Intelligence* 38 (2): 161-198.

Epstein, S. L. 1989a. The Intelligent Novice - Learning to Play Better. In *Heuristic Programming in Artificial Intelligence - The First Computer Olympiad*, ed. D. Levy and D. Beal. New York: John Wiley.

Epstein, S. L. 1989b. Mediation among Advisors In The *Proceedings of the AAAI Spring Symposium Series on AI and Limited Rationality*, 35-39.

Epstein, S. L. 1990. Deep Forks in Strategic Maps. To appear. Also available as Technical Report, TRS-89-04, Hunter College of the City University of New York.

Fikes, R. E. and Nilsson, N. J. 1972. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2: 189-208.

King, P. F. 1970. A Computer Program for Playing Positional Games. M. S. diss., Case Western Reserve University.

Koffman, E. B. 1968. Learning through Pattern Recognition Applied to a Class of Games. *IEEE Trans. Sys. Sci Cybernetics*, SSC-4 .

Korf, R. E. 1985. Macro-Operators: A Weak Method for Learning. *Artificial Intelligence* 26 (1): 35-77.

Laird, J., Rosenbloom, P. and Newell, A. 1986. Chunking in SOAR: An Anatomy of a General Learning Mechanism. *Machine Learning* 1 (1): 11-46.

Langley, P. 1985. Learning to Search: From Weak Methods to Domain-Specific Heuristics. *Cognitive Science* 9 (217-260.

Lee, K. F. and Mahajan, S. 1988. A Pattern Classification Approach to Evaluation Function Learning. *Artificial Intelligence* 36 (1): 1-26.

Minton, S. 1984. Constraint-Based Generalization - Learning Game-Playing Plans from Single Examples. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, 251-254. Los Altos: William Kaufmann.

Minton, S. 1988. *Learning Search Control Knowledge - An Explanation-Based Approach.* Boston: Kluwer Academic.

Mitchell, T. M., Utgoff, P. E. and Banerji, R. 1983. Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics. In *Machine Learning: An Artificial Intelligence Approach*, ed. R. S. Michalski, J. G. Carbonell and T. M. Mitchell. Palo Alto: Tioga Publishing.

Mooney, R. J. 1988. Generalizing the Order of Operators in Macro-Operators. In *Proceedings of the Fifth International Conference on Machine Learning*, 270-283. Morgan Kaufmann.

Paul, J. L. 1978. Tic-Tac-Toe in n-Dimensions. *Mathematics Magazine* 51: 45-49.

Rosenbloom, P. S. 1982. A World-Championship Level Othello Program. *Artificial Intelligence* 19 (3): 279-320.

Samuel, A. L. 1967. Some Studies in Machine Learning Using the Game of Checkers. II - Recent Progress. *IBM Journal of Research and Development* 11 (6): 601-617.

Schaeffer, J. 1988. Learning from (Other's) Experience. In Proceedings on Computer Game Playing, 51-53. *AAAI 1988 Spring Symposium Series.*

Explanations of Empirically Derived Reactive Plans

Diana F. Gordon (gordon@aic.nrl.navy.mil) **John J. Grefenstette (gref@aic.nrl.navy.mil)**

Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory, Code 5514
Washington, D.C. 20375-5000

Abstract

Given an adequate simulation model of the task environment and payoff function that measures the quality of partially successful plans, competition-based heuristics such as genetic algorithms can develop high performance reactive rules for interesting sequential decision tasks. We have previously described an implemented system, called SAMUEL, for learning reactive plans and have shown that the system can successfully learn rules for a laboratory scale tactical problem. In this paper, we describe a method for deriving explanations to justify the success of such empirically derived rule sets. The method consists of inferring plausible subgoals and then explaining how the reactive rules trigger a sequence of actions (i.e., a strategy) to satisfy the subgoals.

1 Introduction

This report is part of an on-going study concerning learning reactive plans for sequential decision tasks given a simulation of the task environment. In particular, we have been investigating techniques that allow a learning system to actively explore alternative behaviors in simulation, and to construct high performance rules from this experience using competition-based methods. Our current research focuses on learning reactive rules for a variety of tactical scenarios. Learning tactical rules is especially difficult if the environment is only partially modeled, contains other independent agents, or permits only limited sensing of important state variables. Such features reduce the utility of traditional projective problem solving (Mitchell, 1983; Minton et. al, 1989) and favor the use of reactive control rules that respond to current information and suggest useful actions (Agre and Chapman, 1987; Schoppers, 1987).

We have been investigating the usefulness of genetic algorithms and other competition-based heuristics (Grefenstette, 1988) to learn high performance reactive rules in the absence of a strong domain theory. The approach has been implemented in a system called SAMUEL (Grefenstette, 1989). One of the important differences between SAMUEL and many other genetic learning systems is that SAMUEL learns rules expressed in a high level rule language. The use of a symbolic rule language is intended to facilitate the incorporation of more powerful learning methods into the system where appropriate. In this paper, we investigate the use of explanation-based learning methods to explain the success of the empirically learned plans found by the genetic learning system, and to suggest possible improvements.

SAMUEL consists of three major components: a problem specific module, a performance module, and a learning module. The problem specific module consists of the task environment simulation, or world model, and its interfaces. The performance module consists of a competition-based production system that performs matching, conflict resolution and credit assignment. The learning module uses a genetic algorithm to develop high performance reactive plans, each plan expressed as a set of condition-action rules. Each plan is evaluated by testing its performance in controlling the world model through the performance module. Genetic operators, such as crossover and mutation, produce plausible new plans from high performance precursors.

Experiments have shown that SAMUEL learns highly effective reactive plans for laboratory scale tactical problems (Grefenstette, 1989). However, even though the individual rules of a plan can be interpreted, the strategy underlying the plan is often not apparent. We are currently expanding our focus to include the derivation of explanations of SAMUEL's reactive rules. These explanations are expected to clarify the system's performance to system users as well as to generate new reactive rules for SAMUEL.

In this paper, we first discuss a simulated environment to which SAMUEL has been successfully applied. The remainder of the paper is devoted to describing our research on the topic of generating explanations of reactive plans.

This work is part of an on-going study of genetic algorithms for learning tactical plans. The current system is detailed in (Grefenstette, Ramsey & Schultz, 1990). An analysis of the credit assignment methods in appears in (Grefenstette, 1988). A study of the effects of sensor noise on appears in (Schultz, Ramsey & Grefenstette, 1990).

2 The Evasive Maneuvers Problem

We have tested SAMUEL initially in the context of a particular task called Evasive Maneuvers (EM), inspired in part by (Erickson and Zytkow, 1988). In the EM simulation, there are two objects of interest, a plane and a missile, which maneuver in a two-dimensional world. The object is to control the turning rate of the plane to avoid being hit by the approaching missile. The missile tracks the motion of the plane and steers toward the plane's anticipated position. The initial speed of the missile is greater than that of the plane, but the missile loses speed as it maneuvers. If the missile speed drops below some threshold, it loses maneuverability and drops out of the sky. It is assumed that the plane is more maneuverable than the missile, that is, the plane has a smaller turning radius.

There exist six sensors that provide information about the current tactical state:

1) *last-turn*: the current turning rate of the plane. This sensor can assume nine values, ranging from -180 degrees to 180 degrees in 45 degree increments.
2) *time*: a clock that indicates time since detection of the missile. Assumes integer values between 0 and 19.
3) *range*: the missile's current distance from the plane. Assumes values from 0 to 1500 in increments of 100.
4) *bearing*: the direction from the plane to the missile. Assumes integer values from 1 to 12. The bearing is expressed in "clock terminology", in which *12 o'clock* denotes dead ahead of the plane, and *6 o'clock* denotes directly behind the plane.
5) *heading*: the missile's direction relative to the plane. Assumes values from 0 to 350 in increments of 10 degrees. A heading of 0 indicates that the missile is aimed directly at the plane's current position, whereas a heading of 180 means the missile is aimed directly away from the plane.
6) *speed*: the missile's current speed measured relative to the ground. Assumes values from 0 to 1000 in increments of 50.

In addition to the sensors, there is one control variable, namely, the plane's *turning-rate*. Turning-rate has nine possible values, between -180 and 180 degrees in 45 degree increments. The learning objective is to develop a set of decision rules that map current sensor readings into actions that successfully evade the missile whenever possible. The rule condition contains sensor ranges (which may be cyclic), and the action specifies a setting for the control variable. An example of an actual decision rule learning by SAMUEL is the following:

```
RULE 16:
IF      (and (last-turn [-135, 135]) (time [2, 12])
             (range [0, 700]) (bearing [2, 6])
             (heading [0, 30]) (speed [100, 950]))
THEN    (turn 90)
STRENGTH    949
```

The EM process is divided into episodes that begin with the missile approaching the plane from a randomly chosen direction and that end when either the plane is hit or the missile velocity falls below a given threshold. The critic module provides numeric feedback at the end of each episode that measures the extent to which the missile has been successfully evaded. In the case of unsuccessful evasion, partial credit is given reflecting the plane's survival time (see (Grefenstette et. al, 1990)). Each decision rule is assigned a numeric *strength* that serves as a prediction of the rule's utility. The system uses incremental credit assignment methods (Grefenstette, 1988) to update the rule strengths based on feedback from the critic received at the end of the episode. Experiments have shown that SAMUEL can learn high-performance rule sets (plans) for this task (Grefenstette, 1989).

As can be seen from the above example, while the rules are individually understandable, the underlying strategy behind the rules is not usually clear from inspection. On the other hand, a person who watches a display of the EM task under the control of the learned rules can usually describe the strategy being followed in conceptual terms, for example:

> *Get the missile directly behind the plane, let it get fairly close, then make a hard left turn.*

Once such a description has been obtained,

qualitative reasoning can be applied to explain and justify the strategy. It is expected that explanation-based methods will help to explicate the higher-level strategies being learned, making the results of the empirically learning more easily accepted by human operators and, ultimately, expediting the learning process itself. The remainder of the paper offers initial steps in this direction.

3 Explaining Empirically Derived Rules

Our approach to applying explanation-based techniques to reactive plans can be divided into four phases:

(1) inferring plausible subgoals;
(2) confirming subgoal satisfaction;
(3) creating explanations for reactive plans; and
(4) deriving new rules.

The following sections elaborate our approach to each of the first three phases. The fourth phase is outlined under our plans for future research.

3.1 Inferring Plausible Subgoals

Prior to deriving explanations that SAMUEL's actions are intended to satisfy particular subgoals, the system first attempts to derive plausible subgoals, such as "increase range to missile" or "increase missile deceleration" from a trace of the behavior of the system under the control of the learned rules. A trace covering the actions occurring over a single episode is examined. Traces consist of snapshots of sensor readings followed by the decision rule that has fired. Each snapshot is associated with a time, or state. An example of a trace is shown in Figure 1, where "lturn", "brng", and "hdng" are abbreviations for last turn, bearing, and heading. The action is the turn taken by the plane at this time. In order to simplify the trace shown here, the decision rules do not appear.

A domain theory has been developed for automating subgoal derivation. This part of the domain theory consists of *plausible subgoal derivation* (PSD) rules such as the following:

PSD 1: IF range(m) > RANGE1
THEN PLAUSIBLE-SUBGOAL
(INCREASING deceleration(m))

PSD 2: IF range(m) < RANGE2
THEN PLAUSIBLE-SUBGOAL

lturn	time	range	brng	hdng	speed	action
0	0	1000	7	0	700	0
0	1	600	7	0	650	135
135	2	0	9	350	550	0
0	3	300	3	290	400	45
45	4	200	6	0	300	-135
-135	5	100	4	20	250	90
90	6	100	7	0	200	0
0	7	300	6	0	200	45
45	8	400	7	0	150	45
45	9	500	8	0	150	45
45	10	500	8	0	100	-90
-90	11	600	5	0	100	-45
-45	12	700	4	0	100	45

Fig. 1. Example execution trace.

(INCREASING range(m))

where RANGE1 and RANGE2 are user-definable parameters and m represents the missile. The trace is examined to find the first time at which a PSD rule precondition, such as "range(m) > RANGE1", holds.

The algorithm for finding plausible subgoals is the following:

PSD ALGORITHM: Find the set of all time intervals in the execution trace of an episode for which the sensor values satisfy the PSD rule condition during that interval. This set, called the *trigger set*, consists of situations that would plausibly trigger the implementation of a strategy to satisfy the subgoal specified in the PSD rule.

In the example trace above, if RANGE1 were set to 900, then there is one time interval (of length one unit) that satisfies the condition for PSD1. This interval is [0,0], therefore, the trigger set is simply { [0,0] }. If PSD1 is satisfied, its subgoal, namely, "(INCREASING deceleration(m))", is proposed as a candidate subgoal.

Once a plausible subgoal is found, the next task is to determine whether the subgoal has been satisfied. Satisfaction is determined by applying the confirmation procedure described in the next section for time intervals in the trigger set until either the set of intervals is exhausted or the subgoal has been confirmed.

3.2 Confirming Subgoal Satisfaction

Subgoal satisfaction is determined by once again scanning the execution trace. Scanning begins at the time in the trace following a time interval from the trigger set. Subgoal confirmation requires an additional domain theory. In this case, SAMUEL's decision rule language is extended to capture further information from the trace. For example, the system extracts from the trace information about the *change* in sensor values over time. The speed or range of the missile, for instance, may increase from one state to the next. By scanning the trace over multiple states, the system derives acceleration and range increase information for confirming subgoal satisfaction.

The confirmation of subgoal satisfaction begins when a time interval is chosen from the trigger set. In the current implementation, the user defines a window over which the subgoal satisfaction check is executed. The window begins at a user-defined time that is after the trigger set time interval. Continuing with the example above, suppose the system must confirm that the increasing missile deceleration goal has been achieved over the time window that extends from time 1 to time 3. Then the change in missile speed over this interval is checked to be certain that missile deceleration is increasing. The deceleration is increasing from 100 to 150 over this time interval. Therefore, subgoal satisfaction has been confirmed.

Once subgoals have been derived and confirmed, explanations may be generated to justify the observed behavior. The next section describes the process of explanation generation.

3.3 Creating Explanations

After deriving plausible subgoals and confirming that they are satisfied, explanations may be formed which prove that sequences of SAMUEL's decision rules satisfy the subgoals. Explaining failure to satisfy subgoals is presented as future work.

Creating justifications for successful subgoal satisfaction requires the development of a domain theory that captures important results of particular actions. We are adapting Forbus's Qualitative Process Theory (Forbus, 1984) for the interpretation of the empirically derived rules similarly to the way this theory is adapted in (Gervasio, 1989). Qualitative Process Theory (QP Theory) expresses common sense notions about qualitative relationships between objects.

We are currently using QP Theory to define *processes* relevant to EM. A process is defined in (Forbus, 1984) as something that acts through time to change the parameters of objects in a situation. Example processes are fluid and heat flow, boiling, and motion. We define an EM process below. The individuals are the objects on which the process acts. The quantity conditions are inequalities regarding the quantities of individuals that can be predicted solely within dynamics. Preconditions are conditions that must hold during the process but which need not be predictable using dynamics. Relations are statements that are true during the process. A process is *active* whenever its preconditions and quantity conditions hold. The Q+/Q- relations define qualitative proportionalities. (Q+ *X*, *Y*) means that parameter *X* is directly proportional to parameter *Y*. (Q- *X*, *Y*) means that *X* and *Y* are inversely proportional.

process missile-evasion (p, m)

Individuals:
p, a plane
m, a missile

Quantity Conditions:
speed(*p*) > 0
speed(*m*) > 0

Preconditions:
range(*m*) > 0

Relations:
(Q+ deceleration(*m*), turning-rate(*m*))
(Q+ turning-rate(*m*), turning-rate(*p*))
(Q- speed(*p*), turning-rate(*p*))
(Q- turning-rate(*m*), range(*m*))

The above process description is incomplete and is not entirely accurate. Since we do not intend to engineer a complete and perfect domain theory, our system will eventually possess a capability to diagnose errors in its domain theory.

Once a partial domain theory exists, it is possible to create plausible explanations of the events that occurred during an EM episode. Explanations are derived by creating proofs using the process relations similarly to (Gervasio, 1989). The proof begins with an observable but noncontrollable subgoal and terminates when a change in a controllable parameter has been found that is believed to have caused

subgoal satisfaction. The body of the proof consists of QP Theory relational rules, such as those presented above. For example, the following proof explains how the increasing turning-rate of the plane eventually causes the missile deceleration to increase.

(EXPLANATION
 (INCREASING deceleration(*m*))
 ((Q+ deceleration(*m*) turning-rate(*m*))
 (Q+ turning-rate(*m*) turning-rate(*p*))
 (INCREASING turning-rate(*p*))))

The above proof has terminated with a statement that the plane turning rate is increasing. (The plane turning rate is currently the only controllable parameter.) The increasing turning rate is hypothesized as having initiated a strategy to achieve subgoal satisfaction. The system next verifies (by examining the execution trace) that this behavior has, in fact, occurred. For the above example, this would consist of a check to be certain that the plane turning rate is increasing during the time period that begins during the trigger set time interval and ends at some user-specified time following this interval. In the example trace above, the condition that the turning rate must be increasing would be satisfied if the plane's actions were examined from time 0 to time 1.

The selection of times for checking both subgoal satisfaction and triggering behaviors is currently done by the user. These are important parameters, yet they are difficult to choose. We next describe our plans for future work. These plans include automating the choice of these parameters, as well as other parts of the system.

4 Future Work

There are a few important directions that we plan to pursue. The first direction consists of ordering explanations according to their degree of plausibility. The second direction consists of using the explanations to generate new decision rules for SAMUEL. Third, we plan to automate the generation of system parameters and rules. The fourth future direction consists of diagnosing failures. Finally, we would like to increase the complexity of the EM problem.

Currently, we are running experiments to determine the differences in the degree of plausibility of various explanations. The manner in which this is being done is by generating explanations from multiple episode traces. From our experiences with explanation generation, we have been observing that some explanations/subgoals are considered plausible more frequently than others. We plan to use this information about the frequency to order the PSD rules in a manner that reflects the plausibility of explanations, e.g., more plausible subgoals are tried first.

The second direction for future research consists of generating new decision rules from the explanations. If a subgoal is satisfied, and an explanation is generated for subgoal satisfaction, then the system can generalize the explanation (perhaps using the explanation-based learning methods of (Mitchell, Keller & Kedar-Cabelli, 1986)) and then use the generalized explanation to generate new decision rules. Given a successful explanation, SAMUEL's performance can benefit by the creation of new decision rules that are expected to achieve the same results as the rules from which the explanation is formed. The process of generating decision rules from generalized explanations is one of rule specialization. We are currently considering using ideas from MARVIN (Sammut and Banerji, 1986) for designing the rule specialization process. Once new decision rules have been created, they can be fed back into SAMUEL's performance module to augment the existing rule sets. These modified rule sets may then be empirically evaluated using the EM simulator.

The third direction planned for our research is the automation of certain portions of the system that are currently provided by the user. For example, system parameters, such as the user-input window size for subgoal confirmation, might be empirically determined. Furthermore, the domain theory might also be derived empirically. For instance, the Q+/- relationships in the domain theory for explanations could be extracted from the execution traces.

Although we have been able to generate explanations for successful subgoal satisfaction, a ripe area for future research is the addition of the ability to handle failures. If the system derives an explanation that the reactive rules are intended to achieve a particular subgoal, but the trace does not verify that the subgoal has been satisfied, then there exist four possible cases:

(1) The chosen explanation is incorrect, but the domain theory is not faulty
(2) The plausible subgoal that is inferred is not actually the subgoal that the system is trying to achieve
(3) The reactive rules are intended to achieve a

subgoal, but the system has encountered some unexpected interference
(4) The domain theory is incorrect or incomplete

Although the generation of alternative explanations would be a relatively simple a solution for the first case, the other cases would require more sophisticated error diagnosis.

A final direction for future research is to increase the complexity of the EM problem. For example, the only controllable parameter currently implemented is the plane turning rate. More controllable parameters might be added. Furthermore, the problem difficulty would be greatly increased if the number of missiles were increased. Ultimately, we would like SAMUEL to be able to handle realistic problems.

5 Summary

Progress in generating and using explanations of reactive plans for SAMUEL is expected to provide an important step toward reducing the burden placed on the system's empirical learning mechanisms. The eventual goal of our research is to use these explanations to create high performance reactive plans.

References

Agre, P. and Chapman, D. (1987). Pengi: An implementation of a theory of activity. *Proceedings of the Sixth National Conference on Artificial Intelligence.*

Erickson, M. and Zytkow, J. (1988). Utilizing experience for improving the tactical manager. *Proceedings of the Fifth International Conference on Machine Learning.* Ann Arbor, MI.

Forbus, K. (1984). Qualitative process theory. *Artificial Intelligence,* 24(1-3). North-Holland Publishing Company, Amsterdam, The Netherlands.

Gervasio, M. and DeJong, G. (1989). Explanation-based learning of reactive operators. *Proceedings of the Sixth International Workshop on Machine Learning.* Ithaca, NY. Morgan Kaufmann Publishers, Inc.

Grefenstette, J. (1988). Credit assignment in rule discovery system based on genetic algorithms. *Machine Learning,* 3(2/3). Kluwer Academic Publishers, Hingham, MA.

Grefenstette, J. (1989). A system for learning control strategies with genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms.* Fairfax, VA: Morgan Kaufmann.

Grefenstette, J., Ramsey, C. and Schultz, A. (1990). Learning sequential decision rules using simulation models and competition. To appear in *Machine Learning Journal.* Kluwer Academic Publishers, Hingham, MA.

Minton, S., Carbonell, J., Knoblock, C., Kuokka, D., Etzioni, O., and Gil, Y. (1989). Explanation-based learning: A problem-solving perspective. Carnegie-Mellon University Technical Report Number CMU-CS-89-103.

Mitchell, T. (1983). Learning by experimentation: Acquiring and refining problem-solving heuristics. In R. Michalski, J. Carbonell, and T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach* (Vol. 1). Tioga Publishing Co., Palo Alto, CA.

Mitchell, T., Keller, R. and Kedar-Cabelli, S. (1985). Explanation-based generalization: A unifying view. *Machine Learning,* 1(1). Kluwer Academic Publishers, Hingham, MA.

Sammut, C. and Banerji, R. (1986). Learning concepts by asking questions. In R. Michalski, J. Carbonell, and T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach* (Vol. 2). Morgan Kaufmann Publishers, Los Altos, CA.

Schoppers, M. (1987). Universal plans for reactive robots in unpredictable environments. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence.*

Schultz, A., Ramsey, C. and Grefenstette, J. (1990). Simulation-assisted learning by competition: Effects of noise differences between training model and target environment. In *Proceedings of the Seventh International Machine Learning Conference.* Austin, TX: Morgan Kaufmann.

Learning and Enforcement: Stabilizing environments to facilitate activity.

Kristian J. Hammond
The University of Chicago
Department of Computer Science
Artificial Intelligence Laboratory
1100 East 58th Street
Chicago, IL 60637

Abstract

One of the basic assumptions of work in machine learning is that the environments in which our learning systems are situated are stable enough to make learning useful. While this assumption is warranted in most domains, much of what we tend to think of as a natural regularity in the world has often been artificially imposed in order to make both learning and planning more tractable. This imposition is usually the product of a long-term manipulation of the physical structure, goals, and operators of these domains in the direction of maximal utility. Often, however, it is the product of an agent imposing stability on a domain in an effort to increase the utility of his own planning and learning. This paper examines the idea of what it would mean for an agent to strategically impose order on a domain in an effort to increase the effectiveness of its own learning. In particular, it outlines an initial taxonomy of classes of stability and presents the strategies for increasing overall stability that are associated with each class. Finally, it outlines the basic learning and planning trade-offs that have to be made when stability is optimized.

1 Stability, change, and enforcement

The world is in flux. Every moment brings a new set of states into being and removes an old set from existence. Every action, every plan we know, introduces change in the form of the goals we are trying to achieve and the side-effects of the actions that we are using to achieve them.

The world is also stable. Facts persist over time. Objects tend to stay where they are placed, actions tend to have the same results, and the basic physics of our environment seems to remain fixed and unchanging. Even our goals tend to stay constant over time.

The possibility of change allows us to act at all. The stability of the world, however, allows us to act intelligently. It is our trust in the stability of the world that allows us to predict the future based on the past and build plans based on experience.

There is a direct relationship between the overall stability of an environment and our ability to predict and plan within it. The greater the stability the more certain our predictions, the more powerful our plans.

As both individuals and as societies, we respond to this by trying to increase the stability of our world. We segment our schedules of work, play and relaxation so that each day will tend to look very much like the last. We organize our homes and workspaces so that objects will be in predictable places. We even organize our habits so that particular conjuncts of goals will tend to arise together. In all aspects of our lives, we make moves to stabilize our different worlds.

Of course, all this is done for a reason. Schedules that remain constant over time improve predictability and provide fixed points that reduce the complexity of projection. Few of us need to reason hard about where we will be from 9 to 5 because we have stabilized our schedules with respect to those hours. Fixed locations for objects reduce the need for inference and enable the execution of plans tuned to particular environments. If your drinking glasses are all kept in one cupboard, you can get a drink of water without ever considering the *real* precondition to the plan that they are in there *now*. Likewise, the clustering of goals into standard conjuncts enables the automatic use of plans that are optimized for those conjuncts. A morning routine is exactly that, a routine that is designed to fit a conjunct of goals that can all be satisfied with a well tuned plan. In general, we force stability on the world, and then *enforce* it, in an effort to improve our ability to function in it.

In this paper, we outline this concept of *enforcement* and discuss the different forms that it takes. We examine the idea of what it would mean for an agent to strategically impose order on a domain in an effort to increase the effectiveness of its own learning. In particular, we outline a basic taxonomy of classes of stability and presents the strategies for increasing overall stability that are associated with each class. We examine its relationship to learning and argue that both learning and enforcement are strategies for building up a correspondence between an agent's mental model of the world and the actual physical reality. We also discuss the learning and planning trade-offs that have to be made when stability is optimized.

2 Planning, learning and enforcement

Over the past five years, research in planning has taken a dramatic change in course. Planning researchers have begun to acknowledge that the world is far too complex and uncertain to allow a planner to exhaustively plan for a set of goals prior to execution (Chapman, 1985). More and more, the study of planning is being cast as the broader study of planning, action, learning, and understanding (Agre and Chapman, 1987, Alterman, 1985, and Hammond, 1989).

The particular cast of this relationship that we have been studying is a view of planning as embedded within a memory-based understanding system (Martin, 1990) connected to the environment (Hammond and Converse, 1990). The power of this approach lies in the fact that it allows us to view the planner's environment as well as its plan selections, decisions, conflict resolutions, and action mediation through the single eye of situation assessment and response. We see this integration of planning, understanding, and action as a model of *agency*, in that we are attempting to capture an architecture for an agent embedded in an environment rather than simply a planner abstracted away from an external world.

This integration of planning with understanding, in particular understanding through the use of episodic memory, also provides us with a powerful tool with which to deal with the problem of learning from both planning and execution. One aspect of this view of agency is that it treats planning as a long-term problem that continues over time. Rather than seeing the problems of planning and action in terms of single instances of goals and their related plans, we see planning as also involving the ongoing process of finding the set of plans and plan modifications that are most useful within the planner's domain. For example, the overall cost of constructing a plan can be thought of as amortized over its repeated reuse, but only if we think of planning as the creation of structures that actually will be saved and reused (Marks, Hammond, and Converse 1989).

Part of this process involves the standard issues of learning. This includes learning particular plans, the features that predict their usefulness, and the conditions under which they should be avoided. Here, the overall goal is to develop an internal model of the plans and inferences that are functional in the actual world by adapting the internal world to match the external reality. Much of our work to date (Hammond, Converse, and Marks, 1988 and Hammond, 1989) has been aimed at this sort of learning in the context of planning and execution. In particular, we have been concerned with learning optimized plans for the recurring conjuncts of goals in a domain as well as those that avoid the typical problems that will tend to arise out of a problem space.

The idea of *enforcement* is a somewhat different approach to the goal of building this functional correspondence between an agent's internal state and the external world. The difference between enforcement and previous approaches to learning from planning lies in its use of techniques to shape and stabilize an environment in an effort to optimize the overall utility of plans that already exist or that have just recently been produced. The goal associated with these techniques is the same as that associated with learning in the context of planning—the development of a set of effective plans that can be applied to satisfy the agent's goals. The path toward this goal, however, is one of shaping the world to fit the agent's plans rather than shaping the agent to fit the world. The idea of enforcement, then, rises out of the observation that the result of a long-term interaction between agent and environment includes an adaptation of the environment as well as an adaptation of the agent.

3 Opportunism and enforcement: An example

Our notion of *enforcement* rises out of our approach to planning as only part the study of agency—most specifically—out of our examination of *opportunistic memory*, in the TRUCKER and RUNNER projects (Hammond 1989). In this work, we looked at the issues involved with indexing blocked goals in memory and reawakening them under conditions which would favor their satisfaction. The idea was to combine planning-time reasoning with execution-time understanding in an effort to obtain efficient recognition of and capitalization of execution-time opportunism.

One of the examples that we examined involves an agent going to the grocery store to pick up a quart of orange juice and recalling that he needs milk as well. We argue that there are two aspects to how an agent should repond to this sort of problem. First, he should attempt to incorporate the plans for the recalled goal into the current execution agenda. Second, he should reason about the likelihood of the recalled goal recurring in conjunction with the goal that was already being acted upon and save the plan for the conjunct of the two goals if they were likely to be conjoined in the future. In a sense, these two steps correspond to fixing the plan and then fixing the planner.

One element of this process that interests us is the notion that the more likely it is that that goals will show up in conjunction with each other, the more useful the plan will be. In this example, the utility of saving and attempting to reuse the plan to buy both the orange juice and the milk is maximized when the two goals are guaranteed to show up in conjunction whenever either of the two recurs. This suggests the idea that one of the steps that an agent could take in improving the utility of his plans would be to force the recurrence of the conjuncts of goals over which these plans are optimized. In terms of the orange juice and milk example, this means making sure that the cycles

of use of each resource are synchronized. This can be done by either changing the actual use of the resources to bring them into synchronization or by changing the amounts purchased such that they would be used up at the same time. In either case, the idea is to alter circumstances in the world such that the long term utility of a plan that already exists is optimized. This is done by stabilizing the world with regard to the relative use of the two resources. This type of enforcement is aimed at controlling what we call RESOURCE CYCLE SYNCHRONIZATION in that its goal is to stabilize the use cycles of multiple resources with respect to one another.

Adjusting the amount of orange juice purchased so makes cycle of use match the cycle of use of the milk. This increases the utility of the plan to buy the two together in three ways: optimization of planning, optimization of indexing, and optimization of execution.

- In terms of planning optimization, the agent now has available a plan for a conjunct of goals that he knows will recur so he never needs to recreate it.

  This means never having to reconstruct the GET-ORANGE-JUICE-AND-MILK plan again.

- And in terms of indexing optimization, the plan can be indexed by each of the elements of the conjunct—rather than by the conjunct itself—thus reducing the complexity of the search for the plan in the presence of the individual goals.

  This means that the plan will be automatically suggested when either the HAVE-MILK goal or the HAVE-ORANGE-JUICE goal arises even when the other element of the goal conjunct does not.

- In terms of execution optimization, the agent can decide to commit to and begin execution of the new plan when either of the two goals arises. It can do this because it is able to predict that the other goal is also present, even if it is not explicitly so.

  This means that the agent can begin to run the GET-ORANGE-JUICE-AND-MILK plan when he notices that he is out of either milk or orange juice without being forced to verify that the other goal is active. In some sense, the agent does not have to check the refrigerator to see if he is out of milk.

One way of viewing enforcement is as an extension of planning itself. As in planning, the conditions that are enforced are fixed in the world using the same sorts of actions that result in the satisfaction of goals. The difference is that the actions associated with enforcement result in changes to the actual structure of a domain.

Likewise, enforcement can be seen as an active cousin of learning. Just as learning techniques in planning are designed to build up an effective set of plans and operators for a domain, enforcement techniques are designed to do so as well. The difference here is that learning attempts to satisfy this goal by changing the learner and enforcement attempts to do so by changing the world.

From either point of view, the notion of enforcement is straightforward. For any plan that is learned, its utility can be maximized if the conditions in the world that favor its use can be guaranteed. If the world is unstable with respect to those conditions, one step that an agent can take to optimize the utility of the plan is to *enforce* that stability by changing some aspect of the world so as to make those conditions prevail in all circumstances under which the plan could be run.

4 Stability and enforcement

While RESOURCE CYCLE SYNCHRONIZATION was one of the first instances of stability we encountered, it is by no means the only kind. In our preliminary examination, we have uncovered six other basic types of stability and related enforcement strategies. Each type of stability, when enforced, increases the utility of existing plans and planning processes with respect to the cost of use, the cost of indexing, the cost of projection and/or the likely applicability of the plans that have been stored.

The question is, is it possible to explicate this taxonomy of stability in a way that would allow a system to actually recognize and enforce the different types? The sections that follow, outline this taxonomy with respect to this question by breaking each type down in terms of the following issues:

- **What types of stability are useful in and of themselves?**
- **Over what goals do they allow optimization?**
- **What strategies can be formed to enforce them?**
- **How can opportunities to apply these enforcement strategies be recognized?**

4.1 Stability of location

The most common type of stability that arises in everyday activity is that of location of commonly used objects. Our drinking glasses end up in the same place every time we do dishes. Our socks are always together in a single drawer. Everything has a place and we enforce everything ending up in its place.

In the RUNNER project, we have already begun to see the utility of this sort of stability in terms of optimizing the reuse of specific plans. RUNNER is functioning in a **breakfast world** in which it has to make a pot of coffee in the morning. Stabilizing the location of objects such as the coffee pot, the beans, and the grinder would allow it to simply reuse existing plans with minimal modification. It also reduces the

need for search for the objects in both the knowledge-base and physical sense of the word. Of course, the way to enforce this sort of stability is to alter plans that make use of these objects so that they end up placing them back where they "belong".

Enforcing STABILITY OF LOCATION, then, serves to optimize a wide range of processing goals. First of all, the fact that an often used object or tool is in a set location reduces the need for any inference or projection concerning the effects of standard plans on the objects or the current locations of objects. Second, it allows plans that rely on the objects locations to be run without explicit checks (e.g., no need to explicitly determine that the glasses are in the cupboard before opening it). Third, it removes the need at execution-time for a literal search for the object.

As we mentioned earlier, The enforcement of this sort of stability requires altering plans that make use of these objects so that they end up placing them back where they "belong".

The final question in terms of STABILITY OF LOCATION, then, is the issue of when to attempt enforcement. As in many instances of standard learning, failure is a good indicator. Here, the problem will take the form of an execution-time failure to actually find an object that is both known to exist and is a object essential to a plan being run. Of course, if many plans make use of the object and each prefers it in a different location, then it will not be useful to attempt to enforce its location.

4.2 Stability of schedule

Another common form of stability involves the construction of standard schedules that persist over time. Eating dinner at the same time every day or having preset meetings that remain stable over time are two examples of this sort of stability. The main advantage of this sort of stability is that it allows for very effective projection in that it provides fixed points that do not have to be reasoned about. In effect, the fixed nature of certain parts of an overall schedule reduces that size of the problem space that has to be searched.

A second advantage it that fixed schedules actually allow greater optimization of the plans that are run within the confines of the stable parts of the schedule. Features of a plan that are linked to time can be removed from consideration if the plan is itself fixed in time. For example, by going into work each day at 8.30, an agent might be able to make use of the traffic report that is on the radio at the half-hour. Because the schedule is stable, however, he doesn't have to actually reason about this as an explicit condition of the plan.

Finally, if the schedule is stabilized with regard to a pre-existing norm, (e.g., always have lunch at noon) coordination between agents is also facilitated.

The enforcement strategy associated with STABILITY OF SCHEDULE is simple: don't break the schedule. Here, of course, we see the first instance of a trade-off between enforcement and planning flexibility. While an enforced schedule allows for optimization of search and execution for recurring goals, it often reduces the flexibility required to incorporate new goals into the preset agenda. As with any heuristic that reduces the combinatorics of a search space, there will be times when an optimal plan is not considered.

It is important to realize that the schedule enforced is optimized over the goals that actually do tend to recur. Thus, an agent who is enforcing this sort of stability is able to deal with regularly occurring events with far greater ease than when it is forced to deal with goals and plans outside of its normal agenda. This sort of trade-off in which commonly occurring problems are easier to solve than less common ones seems to be an essential by-product of stabilizing an environment.

Recognition of opportunities to enforce STABILITY OF SCHEDULE is a fairly difficult problem. There are two basic features that are important. First, an agent must recognize that a goal is going to recur and that a single plan is designed to satisfy it. Second, he must recognize that a particular placement in time is optimal for running the plan. While the first of these is fairly simple, the second feature requires either an extensive projection over different times for running the plan or an opportunistic realization that the plan has been run at a particularly good time at one point.

4.3 Stability of satisfaction

Another type of stability that an agent can enforce is that of the goals that he tends to satisfy in conjunction with each other. For example, people living in apartment buildings tend to check their mail on the way into their apartments. Likewise, many people will stop at a grocery store on the way home from work. In general, people develop habits that cluster goals together into compact plans, even if the goals are themselves unrelated. The reason that the plans are together is more a product of the conditions associated with running the plans than the goals themselves.

An important feature of this sort of stability is that the goals are recurring and that the plan associated with the conjunct is optimized with respect to them. Further, the goals themselves must be on loose cycles and robust with regard to over-satisfaction.

The advantage of this sort of STABILITY OF SATISFACTION is that an optimal plan can be used that is already tuned for the interactions between individual plan steps. Second, it can be run habitually, without regard to the actual presence of the goals themselves. As in the case of STABILITY OF LOCATION in which a plan can be run without explicit checks on the locations of objects, STABILITY OF SATISFACTION allows for the execution of plans aimed at satisfying particular goals, even when the goals are not explicitly checked.

The way to enforce this sort of stability is to associate the plan with a single cue—either a goal or

a feature in the world—and begin execution of that plan whenever the cue arises. In this way, the habitual activity can be started even when all of the goals that it satisfies are not present.

The circumstances that suggest habits that will enforce STABILITY OF SATISFACTION are simple. When an opportunity to satisfy a suspended goal (Hammond, 1989) is encountered, the collection of goals currently being satisfied become candidates for conjunction into a habit. The individual cycles of the goals then determine whether or not they are joined into a single plan.

4.4 Stability of plan use

We often find ourselves using familiar plans to satisfy goals even in the face of wide ranging possibilities. For example, when I travel for conferences, I tend to schedule my flight in to a place as late as I can and plan to leave as late as I can on the last day. This optimizes my time at home and at the conference. It also allows me to plan without knowing anything about the details of the conference schedule. As a result, I have a standard plan that I can run in a wide range of situations without actually planning for them in any detail. It works, because it already deals with the major problems (missing classes at home and important talks at the conference) as part of its structure.

The major advantage here in enforcing the STABILITY OF PLAN USE is that the plan that is used is tuned to avoid the typical interactions that tend to come up. This means, of course, that the plans used in this way must either be the result of deep projection over the possible problems that can come up in a domain or be constructed incrementally. A further advantage is that little search through the space of possible plans for a set of goals needs to be done in that one plan is always selected.

The enforcement here is simply a product of choosing to stay with a single plan in the face of a large space of possibilities. In a sense, this is the idea of having a standard operating procedure for a set of goals.

This sort of stabilization is the most emergent of the different types of stability and enforcement that we have looked at. This is because these sorts of plans are the product of incremental debugging over time. The drive towards selection of plans that have worked in the past is simply part of the overall case-based approach. Of course, by starting each new experience with a set of goals with a plan that is debugged with respect a set of already experienced problems and then debugging it with respect to new problems, an agent is actually constructing the plan that will be optimized for the entire set of interactions that he will encounter.

4.5 Policy

Everyone always carries money. This is because we always need it for a wide variety of specific plans. Of course, the policy decision to always have money on hand is itself a form of enforcement.

The interesting issue here is that there are particular states in the world that act as preconditions to a wide variety of plans. By enforcing these states, we can establish a POLICY that will allow us to run these plans without explicit regard to the state itself. Here again, the advantage is that plans can be run without explicit reference to many of the conditions that must obtain for them to be successful. An agent can actually assume conditions hold, because he has a POLICY that makes them hold.

Enforcement of POLICY requires the generation of specific goals to satisfy the policy state whenever it is violated. In terms of policies such as always having money on hand, this means that the lack of cash on hand will force the generation of a goal to have cash, even when no specific plan that will use that cash is present.

The conditions that suggest the enforcement of policies include noting that many plans make use of the state associated with the policy. This can be recognized through the a process of failure driven learning. In effect, the failure to have cash on hand suggests that it is a good policy to enforce. There are other conditions that must hold as well however. The POLICY state must be relatively inexpensive to maintain and the state should be a useful precondition for a wide range of plans.

4.6 Stability of cues

One effective technique for improving plan performance is to improve the proper activation of a plan rather than improve the plan itself. For example, placing an important paper that needs to be reviewed on his desk before going home, improves the likelihood that an agent will see and read it the next day. Marking calendars and leaving notes serves the same sort of purpose.

One important area of enforcement is related to this use of visible cue in the environment to activate goals that have been suspended in memory. The idea driving this type of enforcement is that an agent can decide on a particular cue that will be established and maintained so as to force the recall of commonly recurring goals. One example of this kind of enforcement of STABILITY OF CUES is leaving a briefcase by the door every night in order to remember to bring it into work. The cue itself remains constant over time. This means that the agent never has to make an effort to recall the goal at execution-time and, because the cue is stabilized, it also never has to reason about what cue to use when the goal is initially suspended.

The advantage of this sort of enforcement is that an agent can depend on the external world to provide a stable cue to remind it of goals that still have to be achieved. This sort of stability is suggested when an agent is faced with repeated failures to recall a

goal and the plan associated with the goal is tied to particular objects or tools in the world.

5 A Model of Agency

Of course, this work on enforcement is only part of an overall effort in the study of *agency* the modeling of a system that is able to understand, plan and perform action in the world. We use the term *agency* rather than *planning* because we are attempting to model a the broader class of tasks including the spawning of goals, selection of plans, and execution of actions. Our process model of agency is based on Martin's DMAP understander as well as its antecedent, Schank's *Dynamic Memory* (1982). DMAP uses a memory organization defined by part/whole and abstraction relationships. Activations from environmentally supplied features are passed up through abstraction links and predictions are passed down through the parts of partially active concepts. Subject to some constraints, when a concept has only some of its parts active, it sends predictions down its other parts. When activations meet existing predictions, the node on which they meet becomes active. Finally, when all of the parts of a concept are activated, the concept itself is activated.

To accommodate action, we have added the notion of PERMISSIONS. PERMISSIONS are handed down the parts of plans to their actions. The only actions that can be executed are those that are PERMITTED by the activation of existing plans. Following McDermott (McDermott, 1978), we have also added POLICIES. POLICIES are statements of ongoing goals of the agent. Sometimes these take the form of maintenance goals, such as "Glasses should be in the cupboard." or "Always have money on hand." The only goals that are actively pursued are those generated out of the interaction between POLICIES and environmental features. We would argue that this is, in fact, the only way in which goals can be generated.

Most of the processing takes the form of recognizing circumstances in the external world as well as the policies, goals and plans of the agent. The recognition is then translated into action through the mediation of PERMISSIONS that are passed to physical as well as mental actions.

Goals, plans, and actions interact as follows:

- Features in the environment interact with POLICIES to spawn goals.

  For example, in RUNNER, the specific goal to HAVE COFFEE is generated when the system recognizes that it is morning. The goal itself rises out of the recognition of this state of affairs in combination with the fact that there is a policy in place to have coffee at certain times of the day.

- Goals and environmental features combine to activate plans already in memory.

  Any new MAKE-COFFEE plan is simply the activation of the sequence of actions associated with the existing MAKE-COFFEE plan in memory. It is recalled by RUNNER when the HAVE-COFFEE goal is active and the system recognizes that it is at home.

- Actions are **permitted** by plans and are associated with the descriptions of the world states appropriate to their performance. Once a set of features has an action associated with it, that set of features (in conjunct rather than as individual elements) is now predicted and can be recognized.

  Filling the coffee pot is **permitted** when the MAKE-COFFEE plan is active; it is associated with the features of the pot being in view and empty. This means not only that the features are now predicted but also that their recognition will trigger the action.

- Actions are specialized by features in the environment and by internal states of the system. As with Firby's RAPs (Firby, 1989), particular states of the world determine particular methods for each general action.

  For example, the specifics of a GRASP would be determined by information taken from the world about the size, shape and location of the object being grasped.

- Action level conflicts are recognized and mediated using the same mechanism that recognizes information about the current state of the world.

  For example, when two actions are active (such as filling the pot and filling the filter), a mediation action selects one of them. During the initial phases of learning a plan, this can in turn be translated into a specialized recognition rule which, in the face of a conflict, will always determine the ordering of the specific actions.

- Finally, suspended goals are associated with the descriptions of the states of the world that are amenable to their satisfaction.

  For example, the goal HAVE-ORANGE-JUICE, if blocked, can be placed in memory, associated with the conjunct of features that will allow its satisfaction, such as being at a store, having money and so forth. Once put into memory, this conjunct of features becomes one of the set that can now be recognized by the agent.

Eventually, RUNNER should also be able to recognize opportunities to interleave plans and to modify plans in response to different types of failures.

6 A Framework for the Study of Agency

We do not see this model as a solution to the problems of planning and action. Instead, we see this as a

framework in which to discuss exactly what an agent needs to know in a changing world. Advantages of this framework include:

1. A unified representation of goals, plans, actions and conflict resolution strategies.
2. Ability to learn through specialization of general techniques.
3. A fully declarative representation that allows for meta-reasoning about the planner's own knowledge base.
4. A simple marker-passing scheme for recognition that is domain — and task — neutral.
5. Provision for the flexible execution of plans in the face of a changing environment.

The basic metaphors of action as permission and recognition, and planning as the construction of descriptions that an agent must recognize prior to action, these fit our intuitions about agency. Under this metaphor, we can view research into agency as the exploration of the situations in the world that are valuable for an agent to recognize and respond to. In particular, we have examined and continue to explore content theories of:

- The conflicts between actions that rise out of resource and time restrictions as well as direct state conflicts and the strategies for resolving them.
- The types of physical failures that block execution and their repairs.
- The types of knowledge-state problems that block planning and their repairs.
- The circumstances that actually give rise to goals in the presence of existing policies.
- The possible ways in which existing plans can be merged into single sequences and the circumstances under which they can be applied.
- The types of reasoning errors that an agent can make and their repairs.
- The trade-offs that an agent has to make in dealing with its own limits.
- The different ways in which a goal can be blocked and the resulting locations in memory where it should be placed.

Our goal is a content theory of agency. The architecture we suggest is simply the vessel for that content. Our notion of enforcement is simply one aspect of the overall content model that comprises our theory of agency.

7 The point

In order to plan at all in an environment, it must at least be stable with respect to its basic physics. In order to reuse plans in any interesting way at all, the environment—including the agent—must be stable with respect to other aspects as well. In particular, it must be stable with regard to the physical structure of the environment, the goals that tend to recur and the times at which events tend to take place.

While many environments have this sort of stability, it is often the product of the intervention of agents attempting to stablize it so as to increase the utility of their own plans. In this paper, we have introduced the idea of how an agent could take a strategic approach to tailoring an environment to its plans and the goals it typicaliy must achieve. The goal of this enforcement parallels the goal of learning—the development of a set of effective plans that can be applied to satisfy the agent's goals. The path toward this goal, however, is one of shaping the world to fit the agent's plans rather than shaping the agent to fit the world.

8 Acknowledgements

This work was supported in part by the Defense Advanced Research Projects Agency, monitored by the Air Force Office of Scientific Research under contract F49620-88-C-0058, and the Office of Naval Research under contract N0014-85-K-010.

9 References

Phil Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth Annual Conference on Artificial Intelligence*, pages 268-72. AAAI , 1987.

R. Alterman. Adaptive planning: refitting old plans to new situations. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, 1985.

D. Chapman. Planning for conjunctive goals. Memo AI-802, AI Lab, MIT, 1985.

K. Hammond. *Case-Based Planning: Viewing Planning as a Memory Task*, volume 1 of *Perspectives in Artificial Intelligence*. Academic Press, 1989.

K. Hammond. Opportunistic memory. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*. IJCAI , 1989.

K. Hammond, T. Converse, and M. Marks. Learning from opportunities: Storing and reusing execution-time optimizations. In *Proceedings of the Seventh Annual Conference on Artificial Intelligence*, pages 536–40. AAAI , 1988.

M. Marks, K. Hammond, and T. Converse. Planning in an open world: A pluralistic approach. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, 1989.

C. Martin. *Direct Memory Access Parsing*. PhD thesis, Yale University Department of Computer Science, 1989.

R. Schank. *Dynamic memory: A theory of learning in computers and people*. Cambridge University Press, 1982

Simulation-Assisted Learning by Competition: Effects of Noise Differences Between Training Model and Target Environment

Connie Loggia Ramsey Alan C. Schultz John J. Grefenstette
Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory
Washington, DC 20375-5000
schultz@aic.nrl.navy.mil

Abstract

The problem of learning decision rules for sequential tasks is addressed, focusing on the problem of learning tactical plans from a simple flight simulator where a plane must avoid a missile. The learning method relies on the notion of competition and employs genetic algorithms to search the space of decision policies. Experiments are presented that address issues arising from differences between the simulation model on which learning occurs and the target environment on which the decision rules are ultimately tested. Specifically, either the model or the target environment may contain noise. These experiments examine the effect of learning tactical plans without noise and then testing the plans in a noisy environment, and the effect of learning plans in a noisy simulator and then testing the plans in a noise-free environment. Empirical results show that, while best result are obtained when the training model closely matches the target environment, using a training environment that is more noisy than the target environment is better than using using a training environment that has less noise than the target environment.

1 Introduction

In response to the knowledge acquisition bottleneck associated with the design of expert systems, research in machine learning attempts to automate the knowledge acquisition process and to broaden the base of accessible sources of knowledge. The choice of an appropriate learning technique depends on the nature of the performance task and the form of available knowledge. If the performance task is classification, and a large number of training examples are available, then inductive learning techniques (Michalski, 1983) can be used to learn classification rules. If there exists an extensive domain theory and a source of expert behavior, then explanation-based methods may be applied (Mitchell et. al, 1985). For many interesting sequential decision tasks, there exists neither a database of examples nor a reliable domain theory. In these cases, one method for manually developing a set of decision rules is to test a hypothetical set of rules against a simulation model of the task environment, and to incrementally modify the decision rules on the basis of the simulated experience. This paper presents some initial efforts toward using machine learning to automate the process of learning sequential tasks with a simulation model.

Sequential decision tasks may be characterized by the following general scenario: A decision making agent interacts with a discrete-time dynamical system in an iterative fashion. At the beginning of each time step, the agent observes a representation of the current state and selects one of a finite set of actions, based on the agent's decision rules. As a result, the dynamical system enters a new state and returns a (perhaps null) payoff. This cycle repeats indefinitely. The objective is to find a set of decision rules that maximizes the expected total payoff.[1] Several sequential decision tasks have been investigated in the machine learning literature, including pole balancing (Selfridge, Sutton & Barto, 1985), gas pipeline control (Goldberg, 1983), and the animat problem (Wilson, 1987). For many interesting problems, including the one considered here, payoff is delayed in the sense that non-null payoff occurs only at the end of an episode that may span several decision steps. In fact, the paradigm is quite broad since it includes any problem solving task by defining the payoff to be positive for any goal state and null for non-goal states (Barto, Sutton & Watkins, 1989).

The experiments described here reflect two important methodological assumptions:

1. Our learning system is designed to continue learning indefinitely.
2. Since learning may require experimenting with decision rules that might occasionally produce unacceptable results if applied to the real world, we assume that hypothetical rules will be evaluated in a simulation model.

[1] If payoff is accumulated over an infinite period, the total payoff is usually defined to be a (finite) time-weighted sum (Barto et. al, 1989).

In this methodology, a set of rules is periodically extracted from the learning system to represent the learning system's current plan. This plan is tested in the target environment, and the resulting performance is plotted in a learning curve. Making a clear distinction between the simulation model used for training and the target environment used for testing suggests a number of experiments that measure the effects of differences between the training model and the target environment. Simulation models have played an important role in earlier machine learning efforts (Goldberg, 1983; Booker, 1982, 1988; Wilson, 1985; Buchanan et al., 1988); however, these works do not address the issue of validation of the simulation model with respect to a real target system. The experiment described here represents a step in this direction. Specifically, differences between the training model (the simulator) and the target environment with respect to noise are examined.

2 The Evasive Maneuvers Problem

The experiments described here concern a particular sequential decision task called the *Evasive Maneuvers (EM)* problem, inspired in part by Erickson and Zytkow (1988). The tactical objective is to maneuver a plane to avoid being hit by an approaching missile. The missile tracks the motion of the plane and steers toward the plane's anticipated position. The initial speed of the missile is greater than that of the plane, but the missile loses speed as it maneuvers. If the missile speed drops below some threshold, it loses maneuverability and drops out of the sky. It is assumed that the plane is more maneuverable than the missile. There are six sensors that provide information about the current tactical state:

1. *last-turn*, the current turning rate of the plane;
2. *time*, a clock that indicates time since detection of the missile;
3. *range*, the missile's current distance from the plane;
4. *bearing*, the direction from the plane to the missile;
5. *heading*, the missile's direction relative to the plane; and
6. *speed*, the missile's current speed measured relative to the ground.

Although other sensors could be used, these sensors represent a minimal, realistic set that might be available from the pilots point of view.

Finally, there is a discrete set of actions available to control the plane. In this study, we consider only actions that specify discrete turning rates for the plane.[2] The learning objective is to develop a *tactical plan*, i.e., a set of decision rules that map current sensor readings into actions that successfully evade the missile whenever possible.

The EM problem is divided into *episodes* that begin when the threatening missile is detected and that end when either the plane is hit or the missile is exhausted.[3] It is assumed that the only feedback provided is a numeric payoff, supplied at the end of each episode, that reflects the quality of the episode with respect to the goal of evading the missile. Maximum payoff is given for successfully evading the missile, and a smaller payoff, based on how long the plane survived, is given for unsuccessful episodes.

The EM problem is clearly a laboratory-scale model of realistic tactical problems. Nevertheless, it includes several features that make it a challenging machine learning problem:

- a weak domain knowledge (e.g., no predictive model of missile);
- incomplete state information provided by discrete (possibly, noisy) sensors;
- a large state space; and, of course,
- delayed payoff.

The following sections present one approach to addressing these challenges.

3 SAMUEL on EM

SAMUEL[4] is a system designed to explore competition-based learning for sequential decision tasks (Grefenstette, 1989). SAMUEL consists of three major components: a problem specific module, a performance module, and a learning module. The problem specific module consists of the task environment simulation, or world model (in this case, the EM model), and its interfaces. The performance module is called CPS (Competitive Production System), a production system that interacts with the world model by reading sensors, setting control variables, and obtaining payoff from a critic. In addition to matching, CPS implements conflict resolution as a competition among rules based on rule strength and performs credit assignment based on payoff (Grefenstette, 1988). The learning module uses a genetic algorithm to develop tactical plans, expressed as a set of condition-action rules. Each plan is evaluated on a number of tasks in the world model. As a result of these evaluations, genetic operators, such as *crossover* and *mutation*, produce plausible new plans from high performance parents. More detailed descriptions of SAMUEL appear in (Grefenstette, 1989; Grefenstette, Ramsey & Schultz, 1990).

[2] The current statement of the problem assumes a two-dimensional world. Future experiments will adopt a three-dimensional model and will address problems with multiple control variables, such as controlling both the direction and the speed of the plane.

[3] For the experiments described here, the missile began each episode at a fixed distance from the plane, traveling toward the plane at a fixed speed. The direction from which the missile approached was selected at random.

[4] SAMUEL stands for Strategy Acquisition Method Using Empirical Learning. The name also honors Art Samuel, one of the pioneers in machine learning.

The design of SAMUEL owes much to Smith's LS-1 system (Smith, 1980), and draws on some ideas from classifier systems (Holland, 1986). In a departure from these earlier genetic learning systems, SAMUEL learns plans consisting of rules expressed in a high level rule language.

The rule language allows four types of sensors:

1. *linear*, where the condition specifies an upper and lower bound for the linear ordered values;
2. *cyclic*, which are like linear, except that the values wrap around from the upper bound to the lower bound;
3. *structures*, where the sensor specifies a list of values, and the condition matches if the sensor's current value occurs in a subtree labeled by one of the values in the list; and
4. *pattern*, where the sensor specifies a pattern over the alphabet {0, 1, #}, as in classifier systems.

In the EM domain, only linear and cyclic type sensors are used. An example of a rule for EM follows:

```
if  (and (last-turn 0 45) (time 4 14) (range 500 1400)
         (heading 330 90) (speed 50 850))
then (and (turn 90))
strength 750
```

Each condition (*if* part) specifies a range over the named sensor, and each action (*then* part) specifies the value for the named control variable. The strength is an estimate of the rule's utility and is used for conflict resolution (Grefenstette, 1988).

The use of a high level language for rules offers several advantages over low level binary pattern languages typically adopted in genetic learning systems (Smith, 1980; Goldberg, 1983). First, it makes it easier to incorporate existing knowledge, whether acquired from experts or by symbolic learning programs. Second, it is easier to transfer the knowledge learned to human operators. Third, it makes it possible to combine empirical methods such as genetic algorithms with analytic learning methods that explain the success of the empirically derived rules (Gordon & Grefenstette, 1990).

4 Evaluation of the Method

This section presents an empirical study of the performance of SAMUEL on the EM problem with respect to the differences between the simulation model in which the knowledge is learned and the target environment in which the learned knowledge will be used.

4.1 Experimental Design

The learning curves shown in this section reflect our assumptions about the methodology of simulation-assisted learning. In particular, we make a distinction between the world model used for learning and the target environment. Let E denote the target environment for which we want to learn decision rules. Let M denote a simulation model of E that can be used for learning. The assumption is that learning continues indefinitely in the background using system M, while the plan being used on E is periodically updated with the current hypothetical plan of the learning system. The genetic algorithm in SAMUEL evaluates the fitness of each plan in its population by measuring its performance on a number of episodes (currently, 10 episodes per plan) in the training model M. A plan's fitness determines its reproductive probability for the next generation. At periodic intervals (currently, 10 generations), a single plan is extracted from the current population to represent the learning system's current hypothetical plan. The extraction is accomplished by re-evaluating the top 20% of the current population on 100 randomly chosen episodes on the simulation model M. The plan with the best performance in this phase is designated the current hypothesis of the learning system. This plan is tested in the environment E for 100 randomly chosen problem episodes. The plots show the sequence of results of testing on E, using the current plans periodically extracted from the learning system. Distinguishing these two systems permits the study of how well the learned plans behave if E varies significantly from M, as is likely in practice.

Because SAMUEL employs probabilistic learning methods, all graphs represent the mean performance over 20 independent runs of the system, each run using a different seed for the random number generator. When two learning curves are plotted on the same graph, a vertical line between the curves indicates that there is a statistically significant difference between the means represented by the respective plots (with significance level $\alpha = 0.05$) at that point on the curves. This device allows the reader to see significant differences between two approaches at various points during the learning process. We feel that this is a better way to compare learning curves for continuously learning systems than, say, running the two systems for a fixed amount of time and comparing the performance of the final plans.

4.2 Sensitivity to Sensor Noise

Noise is an important topic in machine learning research, since real environments can not be expected to behave as nicely as laboratory ones. While there are many aspects of a real environment that are likely to be noisy, we can identify three major sources of noise in the kinds of sequential decision tasks for which SAMUEL is designed: sensor noise, effector noise, and payoff noise. Sensor noise refers to errors in sensor data caused by imperfect sensing devices. For example, if the radar indicates that the range to an object is 1000 meters when in fact the range is 875 meters, the performance system has received noisy data. Effector noise refers to errors arising when effectors fail to perform the action indicated by the current control settings. For example, in the EM world, an effector command might be (turn 45), meaning that the effectors should initiate a 45 degree left turn. If the plane turns at a different rate, the effector has

introduced some noise. An additional source of noise can arise during the learning phase, if the critic gives noisy feedback. For example, a noisy critic might issue a high payoff value for an episode in which the plane is hit.

While all of these types of noise are interesting, we restrict our attention to the noise caused by sensors. To test the effects of sensor noise on SAMUEL, two environments were defined. In one environment, the sensors are noise-free. In the second environment, noise is added to each of the four external sensors that indicate the missile's range, bearing, heading, and speed. Noise consists of a random draw from a normal distribution with mean 0.0 and standard deviation equal to 10% of the legal range for the corresponding sensor. The resulting value is then discretized according to the defined granularity of the sensor. For example, suppose the missile's true heading is 66 degrees. The noise consists of a random draw from a normal distribution with standard deviation 36 (10% of 360 degrees), resulting in a value of, say, 22. The noisy result, 88, is then discretized to the nearest 10 degree boundary (as specified by the granularity of the heading sensor), and the final sensor reading is 90. As this example shows, the amount of noise in this environment is rather substantial.

Given the noisy environment and the noise-free environment, there are four possible experimental conditions, depending on which environment is used for the simulation model (M) and which is used for the target environment (E). For each experimental condition, the genetic algorithm was executed for 200 generations, and the results are shown in Figures 1 and 2.

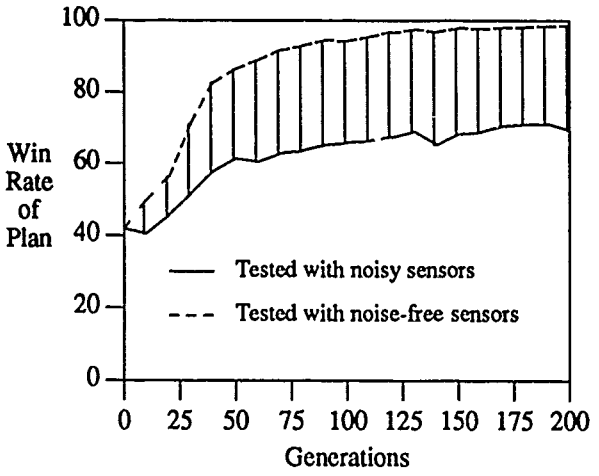


Figure 1: Learning with Noise-Free Sensors

In Figure 1, training was performed in the model with noise-free sensors, and the resulting plans were tested in the environment with noise-free sensors (dashed curve) and noisy sensors (solid curve). In Figure 2, training was performed in the model with noisy sensors, and again tested with both noise-free sensors (dashed curve) and noisy sensors (solid curve).

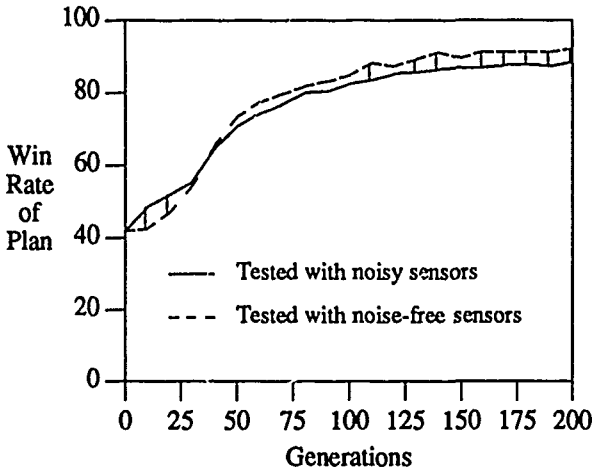


Figure 2: Learning with Noisy Sensors

Figure 1 shows that plans learned with noise-free sensors perform significantly worse throughout the learning period when the testing environment contains noise (solid curve) than when the testing environment is noise-free (dashed curve). For example, after 200 generations, the current tactical plan evades the missile about 98% of the time when the sensors are noise-free, but only about 70% of the time when the sensors are noisy. On the other hand, Figure 2 shows that the plans learned under noisy conditions perform fairly well in both target environments. After 200 generations, the current tactical plan evades the missile about 88% of the time when the sensors are noisy, and about 92% when the sensors are noise-free. Clearly, more robust rules are being learned, at a cost of slower improvement.

By comparing the two dashed curves (or the two solid curves) in Figures 1 and 2, it may be concluded that, for a fixed target environment, SAMUEL learns best when the training environment matches the target environment. However, this ideal case will not generally be realized in practice, especially if the target is a real-world system and the training model is a simulation. Our results show that using a training environment that is less regular (in this case, more noisy) than the target environment is better than having a training model with spurious regularities (e.g., noise-free sensors) that do not occur in the target environment.

5 Summary and Further Research

One important lesson of the empirical study is that SAMUEL is an opportunistic learner, and will tailor the plans that it learns to the regularities it finds in the training model. It follows that the closer the training model matches the conditions expected in the target environment -- in terms of sensor noise -- the better the learned plans will be. In the absence of a perfect match between training model and target environment, it is better to have too little regularity in the training model than too much.

This study illustrates just one of many investigations one might pursue in assessing the effects of differences between the model and the target environment. In another study, we have examined the effect of differences in the initial conditions (i.e., the missile's initial range, speed, and heading) between the training model and the target environment. Preliminary results support the general conclusion reported here -- that it is far less risky to have a training model with overly general initial conditions than to have one with overly restricted initial conditions (Grefenstette et. al, 1990).

Current efforts are also aimed at augmenting the task environment to test SAMUEL's ability to learn tactical plans for more realistic scenarios. Multiple incoming threats will be considered, as well as multiple control variables (e.g., accelerations, directions, weapons, etc.).

As simulation technology improves, it will become possible to provide learning systems with high fidelity simulations of tasks whose complexity or uncertainty precludes the use of traditional knowledge engineering methods. No matter what the degree of sophistication of the simulator, it will be important to assess the effects on any learning method of the differences between the simulation model and the target environment. These initial studies with a simple tactical problem have shown that it is possible for learning systems based on genetic algorithms to effectively search a space of knowledge structures and discover sets of rules that provide high performance in a variety of target environments. Further developments along these lines can be expected to reduce the manual knowledge acquisition effort required to build systems with expert performance on complex sequential decision tasks.

5.0.1 References

Barto, A. G., R. S. Sutton and C. J. C. H. Watkins (1989). Learning and sequential decision making. COINS Technical Report, University of Massachusetts, Amherst.

Booker, L. B. (1982). *Intelligent behavior as as adaptation to the task environment.* Doctoral dissertation, Department of Computer and Communications Sciences, University of Michigan, Ann Arbor.

Buchanan, B. G. J. Sullivan, T. P. Cheng and S. H. Clearwater (1988). Simulation-assisted inductive learning. *Proceedings Seventh National Conference on Artificial Intelligence.* (pp. 552-557).

Erickson, M. D. & J. M. Zytkow (1988). Utilizing experience for improving the tactical manager. *Proceedings of the Fifth International Conference on Machine Learning.* Ann Arbor, MI. (pp. 444-450).

Goldberg, D. E. (1983). *Computer-aided gas pipeline operation using genetic algorithms and machine learning,* Doctoral dissertation, Department Civil Engineering, University of Michigan, Ann Arbor.

Gordon, D. G & J. J. Grefenstette (1990). Explanations of empirically derived reactive plans. *Proceedings of the Seventh International Conference of Machine Learning,* Austin, Texas.

Grefenstette, J. J. (1988). Credit assignment in rule discovery system based on genetic algorithms. *Machine Learning, 3(2/3),* (pp. 225-245).

Grefenstette, J. J. (1989). Incremental learning of control strategies with genetic algorithms *Proceedings of the Sixth International Workshop on Machine Learning.* Ithaca, NY: Morgan Kaufmann. (pp. 340-344).

Grefenstette, J. J., Connie Loggia Ramsey, and Alan C. Schultz (1990). Learning sequential decision rules using simulation models and competition. To appear in *Machine Learning.*

Holland J. H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R.S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach,* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Michalski, R. S. (1983). A theory and methodology for inductive learning. *Artificial Intelligence, 20(2),* (pp. 111-161).

Mitchell, T. M., S. Mahadevan and L. Steinberg (1985). LEAP: A learning apprentice for VLSI design. *Proc. Ninth IJCAI,* (pp. 573-580). Los Angeles: Morgan Kaufmann.

Selfridge, O., R. S. Sutton and A. G. Barto (1985). Training and tracking in robotics. *Proceedings of the Ninth International Conference on Artificial Intelligence.* Los Angeles, CA. August, 1985.

Smith, S. F. (1980). *A learning system based on genetic adaptive algorithms,* Doctoral dissertation, Department of Computer Science, University of Pittsburgh.

Wilson, S. W. (1985). Knowledge growth in an artificial animal. *Proceedings of the International Conference Genetic Algorithms and Their Applications* (pp. 16-23). Pittsburgh, PA.

Wilson, S. W. (1987). Classifier systems and the animat problem. *Machine Learning, 2(3),* (pp. 199-228).

Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming

Richard S. Sutton
GTE Laboratories Incorporated
Waltham, MA 02254
sutton@gte.com

Abstract

This paper extends previous work with Dyna, a class of architectures for intelligent systems based on approximating dynamic programming methods. Dyna architectures integrate trial-and-error (reinforcement) learning and execution-time planning into a single process operating alternately on the world and on a learned model of the world. In this paper, I present and show results for two Dyna architectures. The Dyna-PI architecture is based on dynamic programming's policy iteration method and can be related to existing AI ideas such as evaluation functions and universal plans (reactive systems). Using a navigation task, results are shown for a simple Dyna-PI system that simultaneously learns by trial and error, learns a world model, and plans optimal routes using the evolving world model. The Dyna-Q architecture is based on Watkins's Q-learning, a new kind of reinforcement learning. Dyna-Q uses a less familiar set of data structures than does Dyna-PI, but is arguably simpler to implement and use. We show that Dyna-Q architectures are easy to adapt for use in changing environments.

1 Introduction to Dyna

How should a robot decide what to do? The traditional answer in AI has been that it should deduce its best action in light of its current goals and world model, i.e., that it should *plan*. However, it is now widely recognized that planning's usefulness is limited by its computational complexity and by its dependence on an accurate world model. An alternative approach is to do the planning in advance and compile its result into a set of rapid *reactions*, or situation-action rules, which are then used for real-time decision making. Yet a third approach is to *learn* a good set of reactions by trial and error; this has the advantage of eliminating the dependence on a world model. In this paper I briefly introduce *Dyna*, a class of simple architectures integrating and permitting tradeoffs among these three approaches.

Dyna architectures use machine learning algorithms to approximate the conventional optimal control technique known as *dynamic programming (DP)* (Bellman, 1957; Ross, 1983). DP itself is not a learning method, but rather a computational method for determining optimal behavior given a complete model of the task to be solved. It is very similar to state-space search, but differs in that it is more incremental and never considers actual action *sequences* explicitly, only single actions at a time. This makes DP more amenable to incremental planning at execution time, and also makes it more suitable for stochastic or incompletely modeled environments, as it need not consider the extremely large number of sequences possible in an uncertain environment. Learned world models are likely to be stochastic and uncertain, making DP approaches particularly promising for learning systems. Dyna architectures are those that learn a world model online while using approximations to DP to learn and plan optimal behavior.

Intuitively, Dyna is based on the old idea that planning is like trial-and-error learning from hypothetical experience (Craik, 1943; Dennett, 1978). The theory of Dyna is based on the theory of DP (e.g., Ross, 1983) and on DP's relationship to reinforcement learning (Watkins, 1989, Barto, Sutton & Watkins, 1989, 1990), to temporal-difference learning (Sutton, 1988), and to AI methods for planning and search (Korf, 1990). Werbos (1987) has previously argued for the general idea of building AI systems that approximate dynamic programming, and Whitehead (1989) and others (Sutton & Barto, 1981, Sutton & Pinette, 1985, Rumelhart et al., 1986) have presented results for the specific idea of augmenting a reinforcement learning system with a world model used for planning.

2 Dyna-PI: Dyna by Approximating Policy Iteration

I call the first Dyna architecture *Dyna-PI* because it is based on approximating a DP method known as *policy iteration* (Howard, 1960). The Dyna-PI architecture consists of four components interacting as shown in Figure 1. The *policy* is simply the function formed by the current set of reactions; it receives as input a description of the current state of the world and produces as output an action to be sent to the world. The *world* represents the task to be solved; prototypically it is the robot's external environment. The world receives actions from the policy and produces a next state output and a reward output. The overall task is defined as maximizing the long-term average reward per time step (cf. Russell, 1989). The architecture also includes an explicit *world model*. The world model is intended to mimic the one-step input-output behavior of the real world. Finally, the Dyna-PI architecture includes an *evaluation function* that rapidly maps states to values, much as the policy rapidly maps states to actions. The evaluation function, the policy, and the world model are each updated by separate learning processes.

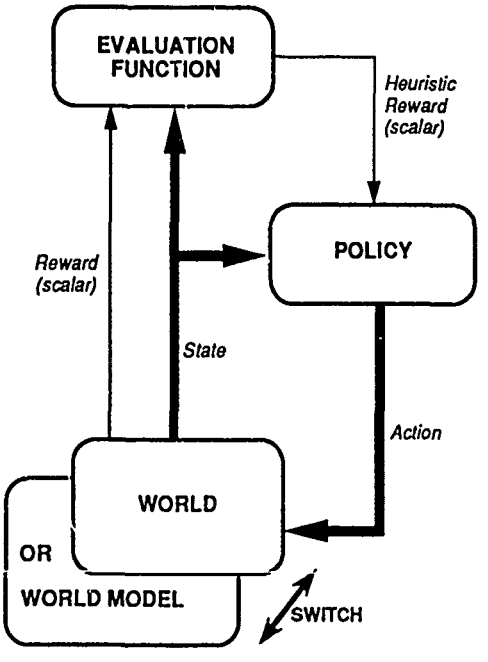


Figure 1: Overview of the Dyna Architecture. With the world in place as shown we have reinforcement learning; with the world model switched in place of the world we have planning.

For a fixed policy, Dyna-PI is simply a reactive system. However, the policy is continually adjusted by an integrated planning/learning process. The policy is, in a sense, a *plan*, but one that is completely conditioned by current input. The planning process is incremental and can be interrupted and resumed at any time. It consists of a series of shallow seaches, each typically of one step ply, and yet ultimately produces the same result as an arbitrarily deep conventional search. I call this *relaxation planning*. Dynamic programming is a special case of this.

Relaxation planning is based on continually adjusting the evaluation function in such a way that credit is propagated to the appropriate steps within action sequences. Generally speaking, the evaluation $e(x)$ of a state x should be equal to the best of the states y that can be reached from it in one action, taking into consideration the reward (or cost) r for that one transition:

$$e(x) \text{ “}=\text{”} \max_{a \in Actions} E\{r + e(y) \mid x, a\}, \qquad (1)$$

where $E\{\cdot \mid \cdot\}$ denotes a conditional expected value and the equal sign is quoted to indicate that this is a condition that we would like to hold, not one that necessarily does hold. If we have a complete model of the world, then the right-hand side can be computed by looking ahead one action. Thus we can generate any number of training examples for the process that learns the evaluation function: for any x, the right-hand side of (1) is the desired output. If the learning process converges such that (1) holds in all states, then the optimal policy is given by choosing the action in each state x that achieves the maximum on the right-hand side. There is an extensive theoretical basis from dynamic programming for algorithms of this type for the special case in which the evaluation function is tabular, with enumerable states and actions. For example, this theory guarantees convergence to a unique evaluation function satisfying (1) and that the corresponding policy is optimal (Ross, 1983).

The evaluation function and policy need not be tables, but can be more compact function approximators such as decision trees, k-d trees, connectionist networks, or symbolic rules. Although the existing theory does not apply to these machine learning algorithms directly, it does provide a theoretical foundation for exploring their use in this way. This kind of planning also extends conventional state-space planning in that it is applicable to stochastic and uncertain worlds and to non-boolean goals.

The above discussion gives the general idea of relaxation planning, but not the exact form used in

policy iteration and Dyna-PI, in which the policy is adapted simultaneously with the evaluation function. The evaluations in this case are not supposed to reflect the value of states given optimal behavior, but rather their value given current behavior (the current policy). As the current policy gradually approaches optimality, the evaluation function also approaches the optimal evaluation function. In addition, Dyna-PI is a *Monte Carlo* or *stochastic approximation* variant of policy iteration, in which the world model is only sampled, not examined directly. Since the real world can also be sampled, by actually taking actions and observing the result, the world can be used in place of the world model in these methods. In this case, the result is not relaxation planning, but a trial-and-error learning process much like reinforcement learning (see Barto, Sutton & Watkins, 1989, 1990). In Dyna-PI, both of these are done at once. The same algorithm is applied both to real experience (resulting in learning) and to hypothetical experience generated by the world model (resulting in relaxation planning). The results in both cases are accumulated in the policy and the evaluation function.

There is insufficient room here to fully justify the algorithm used in Dyna-PI, but it is quite simple and is given in outline form in Figure 2. The algorithm is based on a version of (1) modified to discount later as opposed to immediate reward:

$$e(x) \text{ ``}=\text{''} \max_{a \in Actions} E\{r + \gamma e(y) \mid x, a\}, \quad (2)$$

where γ, $0 \leq \gamma < 1$, is the *discount rate*. Whereas (1) is limited to tasks that end with a clear termination event, such as the finding of a goal state or the end of a board game, (2) can be used for tasks that continue indefinitely, with rewards and/or penalties arriving on each step. Algorithms based on (2) are meant to estimate and maximize the expected value of a discounted sum of future reward:

$$E\left\{\sum_{k=0}^{\infty} \gamma^k r_{k+1} \middle| x\right\},$$

where $r_1, r_2, r_3, \ldots$ is the sequence of future rewards. This is a standard optimization criterion in dynamic programming and Markov decision processes.

3 A Navigation Task

As an illustration of the Dyna-PI architecture, consider the task of navigating the maze shown in the upper right of Figure 3. The maze is a 6 by 9 grid of possible locations or states, one of which is marked as the starting state, "S", and one of which is marked as the goal state, "G". The shaded states act as barriers and cannot be entered. All the other states are distinct and completely distinguishable. From each there are four possible actions: UP, DOWN, RIGHT, and LEFT, which change the state accordingly, except where such a movement would take the take the system into a barrier or outside the maze, in which case the location is not changed. Reward is zero for all transitions except for those into the goal state, for which it is +1. Upon entering the goal state, the system is instantly transported back to the start state to begin the next trial.[1] None of this structure and dynamics is known to the Dyna-PI system a priori.

1. Decide if this will be a real experience or a hypothetical one.
2. Pick a state x. If this is a real experience, use the current state.
3. Choose an action: $a \leftarrow Policy(x)$
4. Do action a; obtain next state y and reward r from world or world model.
5. If this is a real experience, update world model from x, a, y and r.
6. Update evaluation function so that $e(x)$ is more like $r + \gamma e(y)$; this is temporal-difference learning.
7. Update policy—strengthen or weaken the tendency to perform action a in state x according to the error in the evaluation function: $r + \gamma e(y) - e(x)$.
8. Go to Step 1.

Figure 2. Inner Loop of the Dyna-PI Algorithm. These steps are repeatedly continually, sometimes with real experiences, sometimes with hypothetical ones.

In this demonstration, the world was assumed to be deterministic, that is, to be a finite-state automaton, and the world model was implemented simply as next-state and reward tables that were filled in whenever a new state-action pair was experienced (Step 5 of Figure 2). The evaluation function was also implemented as a table and was updated (Step 6) according to the simplest temporal-difference learning method: $e(x) \leftarrow e(x) + \beta(r + \gamma e(y) - e(x))$, where β is a positive learning-rate parameter. The policy was implemented as a table with an entry w_{xa} for every pair of state x and action a. Actions were selected (Step 3) stochastically according to a Boltzmann distribution: $P(a|x) = e^{w_{xa}} / \sum_j e^{w_{xj}}$. The policy was updated (Step 8) according to: $w_{xa} \leftarrow w_{xa} + \alpha(r + \gamma e(y) - e(x))$. For

[1] In fact, the goal state is never entered; the UP action from the state below produces a reward of +1 and sends the system directly to the start state.

hypothetical experiences, states were selected (Step 2) at random uniformly over all states previously encountered. The initial values of the evaluation function $e(x)$ and the policy table entries w_{xa} were all zero; the initial policy was thus a random walk. The world model was initially empty; if a state and action were selected for a hypothetical experience that had never been experienced in reality, then the following steps (Steps 4–7) were simply omitted.

In this instance of the Dyna-PI architecture, real and hypothetical experiences were used alternately (Step 1). For each experience with the real world, k hypothetical experiences were generated with the model. Figure 3 shows learning curves for $k = 0$, $k = 10$, and $k = 100$, each an average over 100 runs. The $k = 0$ case involves no planning; this is a pure trial-and-error learning system entirely analogous to those used in some reinforcement learning systems (Barto, Sutton & Anderson, 1983; Sutton, 1984; Anderson, 1987). Although the length of path taken from start to goal falls dramatically for this case, it falls much *more* rapidly for the cases including hypothetical experiences, showing the benefit of relaxation planning using the learned world model. For $k = 100$, the optimal path was generally found and followed by the fourth trip from start to goal; this is very rapid learning. The parameter values used were $\beta = 0.1$, $\gamma = 0.9$, and $\alpha = 1000$ ($k = 0$) or $\alpha = 10$ ($k = 10$ and $k = 100$). The α values were chosen roughly to give the best performance for each k value.

Figure 4 shows why a Dyna-PI system that includes planning solves this problem so much faster than one that does not. Shown are the policies found by the $k = 0$ and $k = 100$ Dyna-PI systems half-way through the second trial. Without planning ($k = 0$), each trial adds only one additional step to the policy, and so only one step (the last) has been learned so far. With planning, the first trial also learned only one step, but here during the second trial an extensive policy has been developed that by the trial's end will reach almost back to the start state. By the end of the third or fourth trial a complete optimal policy will have been found and perfect performance attained.

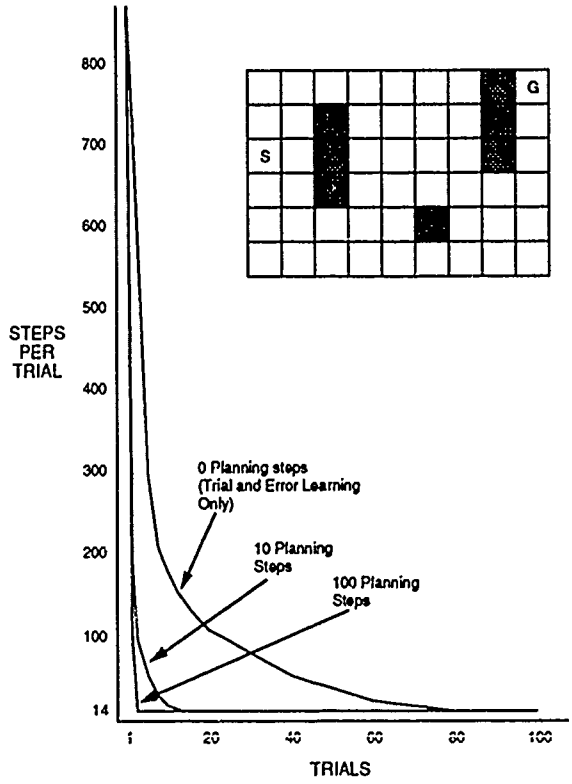


Figure 3. Learning Curves for Dyna-PI Systems on a Simple Navigation Task. A trial is one trip from the start state "S" to the goal state "G". The more hypothetical experiences ("planning steps") using the world model, the faster an optimal path was found.

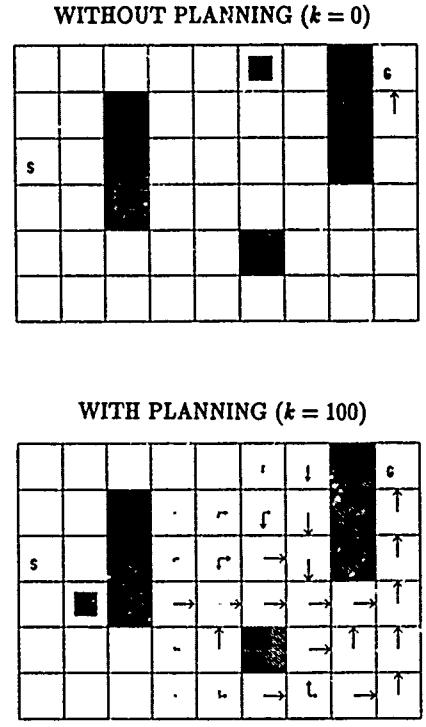


Figure 4. Policies Found by Planning and Non-Planning Dyna-PI Systems by the Middle of the Second Trial. The black square indicates the current location of the Dyna-PI system. The arrows indicate action probabilities (excess over the smallest) for each direction of movement.

4 Problems of Changing Worlds

Suppose that, after a Dyna-PI system has learned the optimal path from start to goal, a new barrier is added that blocks the optimal path. The Dyna-PI system described above will run into the block and then try the formerly effective action many hundreds of times. Eventually, the correct new path may be found, but the process is very slow. It seems inappropriately slow in that the system's world model is updated immediately. Even though the world model knows that the formerly good action is now poor, this is not reflected in the system's behavior for a long time. I call this the *blocking problem.*

Part of the problem is that the alternative actions are never tried, even hypothetically, because the policy assigns them a probability of zero. The model knows these actions are better, but this has no effect unless they are tried. One idea for solving this problem is to allow hypothetical actions to be selected according to a more liberal policy than that used to select real actions. The simplest case of this is that in which hypothetical actions are selected at random uniformly. If this is done, a small adjustment must be made to the evaluation update (Step 6). Recall that the evaluation function is supposed to represent the value of each state given the current policy. If hypothetical are selected uniformly, then the bias toward the current policy must be introduced explicitly. To do this, the evaluation update (Step 6), on hypothetical steps only, is altered to be weighted by the current action probability: $e(x) \leftarrow e(x) + \beta(r + \gamma e(y) - e(x))P(a|x)$. In empirical studies we have indeed found this to be an improvement on the original algorithm, substantially improving the robustness of its convergence onto optimal behavior. However, this does not solve the blocking problem: the system still takes many hundreds of actions into an added barrier before finally finding a way around it.

Now consider a second sort of change in the environment. Suppose, after the optimal path has been learned, a barrier is removed that permits a shorter path from start to goal. The simple Dyna-PI system introduced above is unable to take advantage of such a shortcut, it never wavers from the formerly optimal path and thus never discovers that the former obstacle is gone. I call this the *shortcut problem.* In seeking to improve the Dyna-PI system to handle blocks, we might also seek to improve it to handle shortcuts. What is needed here is some way of continually testing the world model. In the next section we introduce a slightly different architecture that handles both kinds of changes with little increase in complexity.

5 Dyna-Q: Dyna by Q-learning

The Dyna PI architecture is in essence the reinforcement learning architecture that my colleagues and I developed (Sutton, 1984; Barto, Sutton & Anderson, 1983) *plus* the idea of using a learned world model to generate hypothetical experience and to plan. Watkins (1989) subsequently developed the relationships between the reinforcement-learning architecture and dynamic programming (see also Barto, Sutton & Watkins, 1989, 1990) and, moreover, proposed a slightly different kind of reinforcement learning called *Q-learning.* The *Dyna-Q* architecture is the combination of this new kind of learning with the Dyna idea of using a learned world model to generate hypothetical experience and achieve planning.

Whereas the original reinforcement learning architecture maintains two fundamental memory structures, the evaluation function and the policy, Q-learning maintains only one. That one is a cross between an evaluation function and a policy. For each pair of state x and action a, Q-learning maintains an estimate Q_{xa} of the value of taking a in x. The value of a *state* can then be defined as the value of the state's best state-action pair:

$$e(x) \stackrel{\text{def}}{=} \max_a Q_{xa}.$$

In general, the Q-value for a state x and an action a should equal the expected value of the immediate reward r plus the discounted value of the next state y:

$$Q_{xa} \text{ "=" } E\{r + \gamma e(y) \mid x, a\}. \qquad (3)$$

To achieve this goal, the updating steps (Steps 6 and 7 of Figure 2) are implemented by

$$Q_{xa} \leftarrow Q_{xa} + \beta(r + \gamma e(y) - Q_{xa}). \qquad (4)$$

This is the only update rule in Q-learning. We note that it is very similar though not identical to Holland's (1986) bucket brigade and to Sutton's (1988) temporal-difference learning.

So far, the Dyna-Q architecture is slightly simpler than the Dyna-PI architecture. Two data structures have been replaced with one (which is no larger than one of the original two), and one update rule and one parameter (α) have been eliminated. However, Q learning generally requires additional complexity in determining the policy from the Q-values, as we discuss below. One advantage of Q-learning is that it requires no special adjustments if the action selection during hypothetical experience is different from the current policy. Watkins (1989) has shown that the Q-values will converge properly whatever policy is used,

either hypothetically or in reality, as long as all state-action pairs are repetitively tried. In the following experiments, actions were selected at random uniformly (Step 3) on hypothetical experiences.

The simplest way of determining the policy on real experiences is to deterministically select the action that currently looks best—the action with the maximal Q-value. However, as we show below, this approach alone suffers from inadequate exploration and can not solve the shortcut problem. In his work with Q-learning, Watkins implemented the policy probabalistically using a Boltzmann distribution: $P(a|x) = e^{\alpha Q_{xa}} / \sum_j e^{\alpha Q_{xj}}$. An annealing process was added in which α tended to infinity so that even a small difference between Q-values would eventually lead to the best action being selected with probability one. That approach, however, recreates the problem of loss of variability in behavior such that shortcuts can not be found.

To deal directly with the shortcut problem, a new memory structure was added that keeps track of the degree of uncertainty about each component of the model. For each state x and action a, a record is kept of the number of time steps n_{xa} that have elapsed since a was tried in x in a real experience. The square root $\sqrt{n_{xa}}$ is used as a measure of the uncertainty about Q_{xa}.[2] To encourage exploration, each state-action pair is given an *exploration bonus* proportional to this uncertainty measure. For real experiences, the policy is to select the action a that maximizes $Q_{xa} + \epsilon\sqrt{n_{xa}}$, where ϵ is a small positive parameter. This method of encouraging variety is very similar to that used in Kaelbling's (in preparation) interval-estimation algorithm.

However, this approach alone does not take advantage of the planning capability of Dyna architectures. Suppose there is a state-action pair that has not been tested in a long time, but which is far from the currently preferred path, and thus extremely unlikely to be tried even with the exploration bonus discussed above. In a Dyna system, why not expect the system to *plan* an action sequence to go out and test the uncertain state-action pair? If there is genuine uncertainty, then there is potential benefit in going out and trying the action, and thus forming such a plan is simply rational behavior and should be done. It turns out that there is a simple way to do this in Dyna-Q. The exploration bonus of $\epsilon\sqrt{n_{xa}}$ is used not in the policy, but in the update equation for the Q-values. That is, (4) is replaced by:[3]

$$Q_{xa} \leftarrow Q_{xa} + \beta(r + \epsilon\sqrt{n_{xa}} + \gamma e(y) - Q_{xa}). \quad (5)$$

In addition, the system is permitted to hypothetically experience actions is has never before tried, so that the exploration bonus for trying them can be propagated back by relaxation planning. This can be done by starting the system with a non-empty initial model. In the experiments with Dyna-Q systems reported below, actions that had never been tried were assumed to produce zero reward and leave the state unchanged.

6 Changing-World Experiments

Experiments were performed to test the ability of Dyna systems to solve blocking and shortcut problems. Three Dyna systems were used: the Dyna-PI system presented earlier in the paper, a Dyna-Q system including the exploration bonus (5), called the *Dyna-Q+* system,[4] and a Dyna-Q system without the exploration bonus (4), called the *Dyna-Q-* system. All systems used $k = 10$. For the Dyna-PI system, the other parameters were set as in the navigation experiment. For the Dyna-Q systems, they were set at $\beta = 0.5$, $\gamma = 0.9$, and $\epsilon = 0.001$.

The blocking experiment used the two mazes shown in the upper portion of Figure 5. Initially a short path from start to goal was available (first maze). After 1000 time steps, by which time the short path was usually well learned, that path was blocked and a longer path was opened (second maze). Performance under the new condition was measured for 2000 time steps. Average results over 50 runs are shown in Figure 5 for the three Dyna systems. The graph shows a *cumulative* record of the number of rewards received by the system up to each moment in time. In the first 1000 trials, all three Dyna systems found a short route to the goal, though the Dyna-Q+ system did so significantly faster than the other two. After the short path was blocked at 1000 steps, the graph for the Dyna-PI system remains almost flat, indicating that it was unable to obtain further rewards. The Dyna-Q systems, on the other hand, clearly solved the blocking problem, reliably finding the alternate path after about 800 time steps.

[2] The use of the square root is heuristic but not arbitrary, as the standard deviation of all stationary, cumulative random processes increases with the square root of the number of cumulating steps.

[3] Note that this differs from (4) only on hypothetical experiences, as $n_{xa} = 0$ on real experiences.

[4] In these experiments, the Dyna-Q+ system selected the action a in each state x that maximized $Q_{xa} + \epsilon\sqrt{(n_{xa})}$, but we have since found that equally good performance can be obtained simply by picking the action with maximal Q_{xa}.

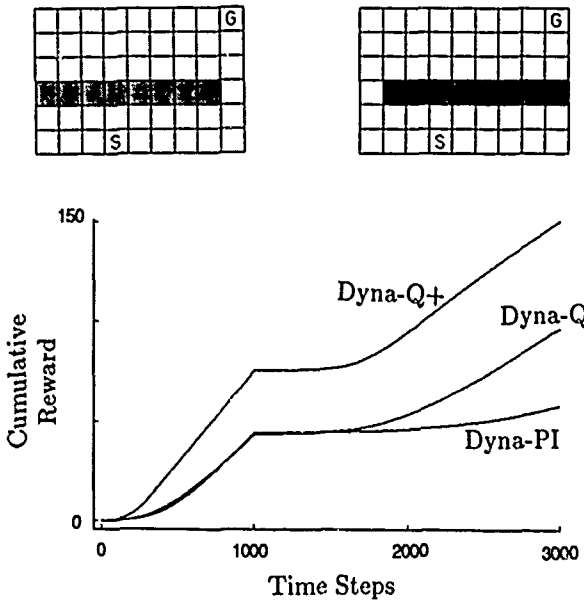


Figure 5. Average Performance of Dyna Systems on a Blocking Task. The left maze was used for the first 1000 time steps, the right maze for the last 2000. Shown is the cumulative reward received by a Dyna system at each time (e.g., a flat period is a period during which no reward was received).

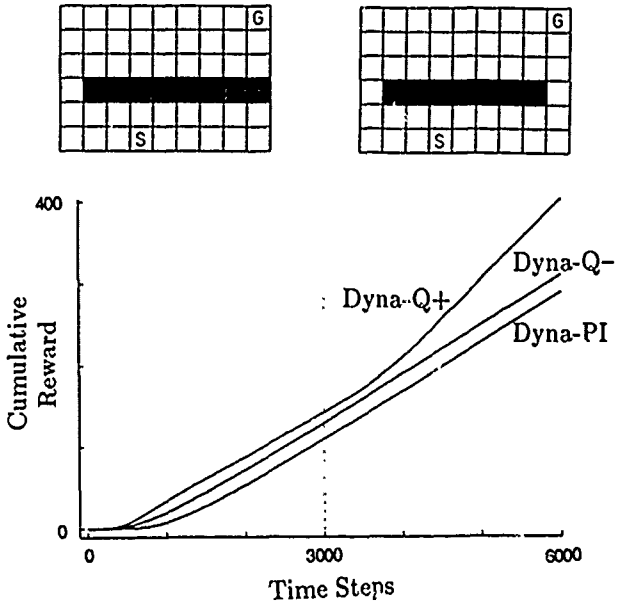


Figure 6. Average Performance of Dyna Systems on a Shortcut Task. The left maze was used for the first 3000 time steps, the right maze for the last 3000. Shown is the cumulative reward received by a Dyna system at each time (e.g., the slope corresponds to the *rate* at which reward was received).

The shortcut experiment began with only a long path available (first maze of Figure 6). After 3000 times steps all three Dyna systems had learned the long path, and then a shortcut was opened without interferring with the long path (second maze of Figure 6). The lower part of Figure 6 shows the results. The increase in the slope of the curve for the Dyna-Q+ system, while the others remain constant, indicates that it alone was able to find the shortcut. The Dyna-Q+ system also learned the original long route faster than the Dyna-Q- system, which in turn learned it faster than the Dyna-PI system. However, the ability of the Dyna-Q+ system to find shortcuts does not come totally for free. Continually re-exploring the world means occasionally making suboptimal actions. If one looks closely at Figure 6, one can see that the Dyna-Q+ system actually acheives a slightly lower *rate* of reinforcement during the first 3000 steps. In a static environment, Dyna-Q+ will eventually perform worse than Dyna-Q-, whereas, in a changing environment, it will be far superior, as here. One possibility is to use a meta-level learning process to adjust the exploration parameter ϵ to match the degree of variability of the environment.

One strength of the Dyna approach is that it applies to stochastic problems as well as deterministic ones. We have explored this direction in recent work, but are not yet ready to present systematic results. The basic idea is to learn a model which predicts not a determinstc next state and next reward, but rather a probability distribution over next states and next rewards. In the simple cases we have explored, this reduces to counting the number of times each possible outcome has occurred. In hypothetical experiences, the expected value on the right of (3) is then estimated using the sample statistics. A slightly different exploration bonus is also needed. Promising preliminary results have so far been obtained for simple problems involving random autonomous agents and stochastic state transitions (e.g., action UP takes the system to the state above 80% of the time, and to a random neighboring state 20% of the time).

Further results are needed for a thorough comparison of Dyna-PI and Dyna-Q architectures, but the results presented here suggest that it is easier to adapt Dyna-Q architectures to changing environments.

7 Limitations and Conclusions

The simple illustrations presented here are clearly limited in many ways. The state and action spaces are small and denumerable, permitting tables to be used for all learning processes and making it feasible for the entire state space to be explicitly explored. For large state spaces it is not practical to use tables or to visit all states; instead one must represent a limited amount of experience compactly and generalize from it. Both Dyna architectures are fully compatible with the use of a wide range of learning methods for doing this.

We have also assumed that the Dyna systems have explicit knowledge of the world's state. In general, states can not be known directly, but must be estimated from the pattern of past interaction with the world (Rivest & Schapire, 1987; Mozer and Bachrach, 1990). Dyna architectures can use state estimates constructed in any way, but will of course be limited by their quality and resolution. A promising area for future work is the combination of Dyna architectures with egocentric or "indexical-functional" state representations (Agre & Chapman, 1987; Whitehead, 1989).

Yet another limitation of the Dyna systems presented here is the trivial form of search control used. Search control in Dyna boils down to the decision of whether to consider hypothetical or real experiences, and of picking the order in which to consider hypothetical experiences. The tasks considered here are so small that search control is unimportant, and thus it was done trivially, but a wide variety of more sophisticated methods could be used. Particularly interesting is the possibility of using Dyna architectures at higher levels to make these decisions.

Finally, the examples presented here are limited in that reward is only non-zero upon termination of a path from start to goal. This makes the problem more like the kind of search problem typically studied in AI, but does not show the full generality of the framework, in which rewards may be received on any step and there need not even exist start or termination states.

Despite these limitations, the results presented here are significant. They show that the use of an internal model can dramatically speed trial-and-error learning processes even on simple problems. Moreover, they show how planning can be done with the incomplete, changing, and oftimes incorrect world models that are contructed through learning. Finally, they show how the functionality of planning can be obtained in a completely incremental manner, and how a planning process can be freely intermixed with reaction and learning processes. I conclude that it is not necessary to choose between planning systems, reactive systems and learning systems. These three can be integrated not only into one system, but into a single algorithm, where each appears as a different facet or different use of that algorithm.

Acknowledgments

The author gratefully acknowledges the extensive contributions to the ideas presented here by Andrew Barto, Chris Watkins and Steve Whitehead. I also wish to also thank the following people for ideas and discussions: Paul Werbos, Luis Almeida, Ron Williams, Glenn Iba, Leslie Kaelbling, John Vittal, Charles Anderson, Bernard Silver, Oliver Selfridge, Judy Franklin, Tom Dean and Chris Matheus.

References

Agre, P. E., & Chapman, D. (1987) Pengi: An implementation of a theory of activity. *Proceedings of AAAI-87*, 268–272.

Anderson, C. W. (1987) Strategy learning with multilayer connectionist representations. *Proceedings of the Fourth International Workshop on Machine Learning*, 103–114. Morgan Kaufmann, Irvine, CA.

Barto, A. G., Sutton R. S., & Anderson, C. W. (1983) Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics 13*: 834–846.

Barto, A. G., Sutton, R. S., & Watkins, C. J. C. H. (1989) Learning and sequential decision making. COINS Technical Report 89-95, Dept. of Computer and Information Science, University of Massachusetts, Amherst, MA 01003. Also to appear in *Learning and Computational Neuroscience*, M. Gabriel and J.W. Moore (Eds.), MIT Press, 1990.

Barto, A. G., Sutton, R. S., & Watkins, C. J. C. H. (1990) Sequential decision problems and neural networks. In *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan Kaufmann, San Mateo, CA.

Bellman, R. E. (1957) *Dynamic Programming.* Princeton University Press, Princeton, NJ.

Craik, K. J. W. (1943) *The Nature of Explanation.* Cambridge University Press, Cambridge, UK.

Dennett, D. C. (1978) Why the law of effect will not go away. In *Brainstorms*, by D. C. Dennett, 71–89, Bradford Books, Montgomery, Vermont.

Howard, R. A. (1960) *Dynamic Programming and Markov Processes.* Wiley, New York.

Kaelbling, L. (in preparation) Learning in Embedded Systems. Stanford Computer Science Ph.D. Dissertation.

Korf, R. E. (1990) Real-Time Heuristic Search. *Artificial Intelligence 42*: 189–211.

Mozer, M. C., & Bachrach, J. (1990) Discovering the structure of a reactive environment by exploration. In *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan Kaufmann, San Mateo, CA. See also Technical Report CU-CS-451-89, Dept. of Computer Science, University of Colorado at Boulder 80309.

Rivest, R. L., & Schapire, R. E. (1987) A new approach to unsupervised learning in deterministic environments. *Proceedings of the Fourth International Workshop on Machine Learning*, 364–375. Morgan Kaufmann, Irvine, CA.

Ross, S. (1983) *Introduction to Stochastic Dynamic Programming.* Academic Press. New York.

Rumelhart, D. E., Smolensky, P., McClelland, J. L., & Hinton, G. E. (1986) Schemata and sequential thought processes in PDP models. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume II*, by J. L. McClelland, D. E. Rumelhart, and the PDP research group, 7–57. MIT Press, Cambridge, MA.

Russell, S. J. (1989) Execution architectures and compilation. *Proceedings of IJCAI-89*, 15–20.

Sutton, R. S. (1984) Temporal credit assignment in reinforcement learning. Doctoral dissertation, Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003.

Sutton, R.S. (1988) Learning to predict by the methods of temporal differences. *Machine Learning 3*: 9–44.

Sutton, R.S., Barto, A.G. (1981) An adaptive network that constructs and uses an internal model of its environment. *Cognition and Brain Theory Quarterly 4*: 217–246.

Sutton, R.S., Pinette, B. (1985) The learning of world models by connectionist networks. *Proceedings of the Seventh Annual Conf. of the Cognitive Science Society*, 54–64. Lawrence Erlbaum, Hillsdale, NJ.

Watkins, C. J. C. H. (1989) *Learning with Delayed Rewards.* PhD thesis, Cambridge University Psychology Department.

Werbos, P. J. (1987) Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions on Systems, Man, and Cybernetics*, Jan-Feb.

Whitehead, S. D. (1989) Scaling reinforcement learning systems. Technical Report 304, Dept. of Computer Science, University of Rochester, Rochester, NY 14627.

ROBOT LEARNING

Reducing Real-world Failures of Approximate Explanation-based Rules

Scott W. Bennett

Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign
405 North Mathews Avenue, Urbana, IL 61801

ABSTRACT

Most current learning and planning systems have been designed to function well in an environment which is a model of the real world. No model of the world can be perfect, however. For a system to actually be able to learn and plan with the real world it must be able to detect problems encountered while acting on the world and to reconcile its model with sensed data. We have constructed an explanation-based learning system called GRASPER which has capabilities to monitor execution of its plans and to tune its model of the world through use of explicit approximations. This paper first characterizes the different kinds of approximations and introduces the use of approximate rules for the purpose of learning uncertainty tolerant plans. Uncertainty tolerant plans offer the important advantage that they can function in spite of errors rather than imposing censors which restrict the generality of a plan. The key issue with uncertainty tolerance approximations is the ability to tune them when the system encounters failures. Our approximations for uncertainty tolerance involve tunable continuous quantities. A new general algorithm is presented which, by creating qualitative representations for the quantitative behavior of an explanation-based rule, can generate explanations as to how to increase the probability of success of the failing expectation through tuning various approximate quantities. A real-world example is given illustrating the tuning process for one of the more common failures occurring with GRASPER operating in the robotics grasping domain. An empirical comparison of failures rates for tuning and non-tuning runs is provided in the task of grasping all pieces to a children's puzzle.

1. Introduction

Most learning and planning systems to date have functioned in isolation from the real world. They work with a simplified representation for the world, usually measuring success by the ability to efficiently produce plans which function well under the assumptions of that representation. This work has produced many invaluable techniques for use in learning and planning. However, one component missing from most of these systems is a mechanism for reconciling their performance in the simplified world with the real world. Any model of real-world behavior will necessarily have discrepancies with how the real world actually behaves. Figure 1 shows the common problem with such systems: an unrealized discrepancy between their model and the real world.

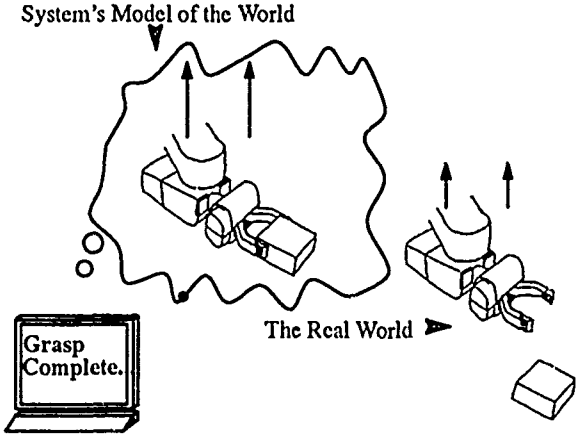


Figure 1. Discrepancies Between a Model and the Real World

Explanation-based learning has shown promise in robotics. In Segre's ARMS system, knowledge about the control of a robotic manipulator in conjunction with geometric object knowledge permitted learning assembly plans from observation [Segre87]. Explanation-based learning permits general plans to be learned from a single example [DeJong86, Mitchell86]. In robotics, a sequence of robot control primitives is used as the example. Attempting to use ARMS to control a real robotic manipulator often resulted in failures due to discrepancies with the real world. This highlighted the need for a mechanism which can adapt a system's model of the world to the real-world environment.

We are currently developing a system called GRASPER to illustrate the use of explanation-based learning in the real world [Bennett89b]. It seeks to learn and execute real-world plans tractably in the presence of uncertainty. Learned plans are therefore error tolerant. Most existing systems which engage in learning from failure impose restrictions on plans when failures are encountered. Recent examples include Hammond's CHEF [Hammond86] and Chien's work on plan refinement [Chien89]. This ensures that the plans won't fail in similar circumstances again but can decrease their overall generality. Plan generality can be retained if plans are learned which function in spite of errors. This is especially important in real-world environments where uncertainty-related errors are so pervasive.

GRASPER is being tested in a robotic grasping domain where it can act on the world through control of a robotic manipulator and can acquire data with a visual system and position and force sensors associated with the manipulator. We are testing system performance at grasping relatively flat pieces from a puzzle for young children. Grasping complex shapes such as these is a difficult ongoing robotics research area and provides an ideal testbed for our learning algorithms. The system employs explicit approximations in the domain theory. It is the tuning of these approximations with experience which is the key to the system. This paper focuses on how approximate rules are tuned over time to increase the uncertainty tolerance of learned plans. First, our approximation terminology is introduced. The approximation tuning algorithm is presented next. The algorithm is then employed on a robotics grasping example. Lastly, we present experimental results comparing tuning algorithm error rates to non–tuning error rates in executing plans on a real–world robotic manipulator.

2. Approximations

An approximation has two important defining features:

Assumability

An approximation must make some statement about the world based not on logical proof but on conjecture.

Tunability

The approximation must provide a method by which it can be tuned as the system acquires new knowledge and/or its goals change. The tuning method should allow adjustment of continuous parameters of the approximation to decrease its error with respect to the true world situation. The tuning method may accept as input new knowledge (obtained from sensor readings) which facilitate this adjustment.

Assumability gives an approximation its efficiency and tractability advantage. This provides a justification that further reasoning need not be done. Tunability indicates that our use of the term approximation requires that they be a function of continuously tunable parameters. By making approximations explicit, rather than implicit as in many AI systems, failures resulting from the approximations can be traced back to inadequate approximations. In much of the work on approximation, the focus is on how to make the approximations initially. This is an important issue in using approximations to improve the efficiency of one's knowledge. Approximations differ from single binary assumptions in that they embody a notion of state and are tuned from their current state, not simply retracted if they lead to a failure as with Chien's approach [Chien89]. Since approximations can be tuned continuously, they also differ from the discrete number of possible adjustments available in fixed abstraction hierarchies like that of Doyle [Doyle86]. But in using approximations to deal with real–world uncertainties, the goal is to improve the uncertainty tolerance of one's knowledge. The unfortunate reality is that everything a system senses from the outside world is in some sense approximate already. To put this in perspective, in the following two subsections, we discuss and distinguish between *data approximations* and *rule approximations*.

2.1. Data Approximations

Data approximations involve representations for measures sensed from the real world. The raw sensor readings obtained by the system are *external approximations*. That is, they can only be tuned through interaction with the world. If a range sensor returned the distance to an object, that distance would be externally approximate. However, one could imagine performing some further actions in the world, such as using tactile sensors to feel first contact with the object, so as to tune that initial approximation for distance to the object. In contrast, an *internal approximation* can be tuned by the system's own reasoning alone without acting on the world. A common type of internal data approximation employed by our system is to reduce the complexity of visually sensed 2–D object contours. Such contours involve hundreds of points and make it difficult and slow to devise grasping points. The internal data approximation currently employed reduces the contour to a n–gon with n much less than the number of sensed contour points. Data approximations, both external and internal, are currently pre–selected for the domain. The tuning of such approximations from their initial values is the key problem here. Our use of internal data approximations is for efficiency not uncertainty tolerance.

2.2. Rule Approximations

Rule approximations are employed when the system plans or understands how a goal can be achieved. They affect the way the system interacts with the world. Consequently, these approximations are useful for building uncertainty tolerance into a plan. They are always internal approximations, capable of being controlled by the system. Approximation techniques, such as those in use by Keller [Keller87], Zweben [Zweben88], and others, which drop rule preconditions, are like our rule approximations but in a discrete, not a continuous, sense. Their approach to improving efficiency of rules through approximations is complementary to ours as both efficiency and uncertainty tolerance are important aspects of a system's real–world performance.[1]

This paper focuses on rule approximations for the purpose of improving the uncertainty tolerance of a plan. Here, the rule approximations involve a choice for continuous parameters whose adaptability to the environment is desired. The initial approximate rules include a declaration of these continuous approximate quantities as well as a set of predicates (antecedents to the approximate rules) which calculate the

1. For a model of operationality for real–world systems which brings together efficiency, uncertainty tolerance and other factors see [Bennett89a].

initial values. These approximate rules are pre–defined as part of the domain knowledge.

We have identified three basic types of rule approximations and employ them all in our current implementation:

controls

These rules directly choose the value for some real–world quantity. For example, in the robotic grasping domain, *chosen–opening–width* is a control approximation which chooses the amount by which the gripper should open for achieving the grasp.

constraints

A constraint rule is used for restricting a choice from among a set of candidates. Each constraint rule functions as part of a multi–constraint rating rule for evaluating a set of candidate choices. In the robotic grasping domain, approximate constraint rules are used in choosing the best faces to use in achieving a grasp. Currently implemented constraint approximations include *opening–width–constraint* and *contact–angle–constraint* for constraining the choice of faces to those with a realizable opening width and a contact angle within the friction angle.

weights

Constraints which are part of one rating rule are combined using weights which are themselves tunable approximations represented by approximate rules. For example, *opening–width–constraint* has an associated weight *opening–width–constraint–weight* used in evaluating it relative to *contact–angle–constraint.*

Next, we present an algorithm for recognizing which approximations, among the declared rule approximations, to tune on failure and how to tune them.

3. A General Tuning Algorithm

Given a goal, the system constructs an explanation for how the goal may be achieved. This can be accomplished in either an understanding mode, given an applied operator sequence, or in a planning mode where the operator sequence is derived. Rules involved in constructing the explanation include approximate rules as outlined above. Most of the facts employed in constructing the explanation are data–approximate having been derived from sensed values from the real world. In order for a monitored action to be achieved in the explanation, a set of expected sensor values must be justified by a further subpart of the explanation. The overall explanation is then generalized using EGGS [Mooney86] and packaged into a rule as with standard EBL systems. When the rule is executed in the real world, those sensors described in the monitored actions are observed. If the sensor readings observed violate the constraints described in the monitored actions, plan execution has failed.[2] In this case, the subpart of the original explanation which justified the expected sensor readings is suspect. Clearly, in the approximate model of the world, no error was foreseen, otherwise the explanation would not have been possible. This suspect subpart of the original explanation is the starting point for our general tuning algorithm.

The tuning algorithm has two major steps:

1) generate a qualitative explanation for how the probability of the failed expectation can be increased through tuning of rule–approximate quantities and

2) perform the actual tuning of the indicated rule approximations.

The key in accomplishing the first step is to express the relationships between generalized variables in the failing subtree as qualitative relations. This will make possible qualitative proofs which relate data–approximate quantities, rule–approximate quantities, and qualitative probabilities of success of the various predicates. The procedure is depicted graphically in Figure 2. First, the sub–tree of the overall explanation which supports the failed expectations is instantiated with the generalized binding list which EGGS produced for the original goal explanation. The predicates at the root and leaves of this sub–tree are asserted in a new situation as qualitative relations. The quantity arguments to these predicates (which are generalized variables) become quantities in our qualitative model of the sub–proof. Any data–approximate or rule–approximate quantities which took part in the original explanation and whose quantity variables are members of the set of generalized quantity variables for the sub–proof are asserted as data–approximate and tunable quantities respectively in the current situation. Once these facts have been asserted which pertain to the specific proof tree, the goal of increasing the probability of the root predicate to the sub–proof can be proved using a set of domain–independent qualitative rules.

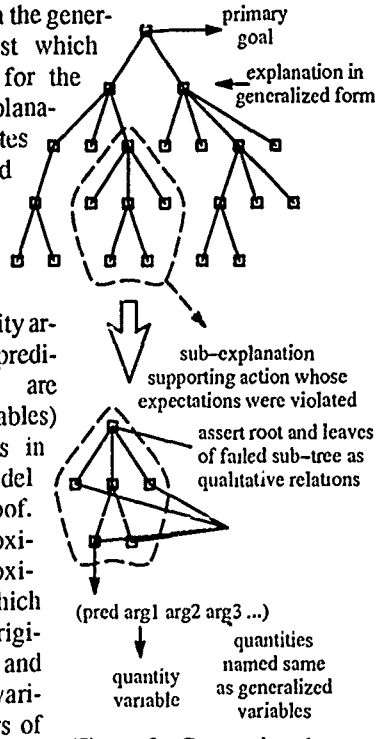


Figure 2. Generating the Qualitative Model

There are four classes of domain independent qualitative rules used by the system for generating the qualitative tuning explanation:

2. Approximation tuning in our system is driven based on *expectation failure*. This idea has long been advocated as a trigger for learning. See [Schank82].

general qualitative inference rules

These are rules for inferring the effects of changes in quantities. For instance, the qualitative proportionality predicate *(Q+ ?a ?b)* asserts that the magnitude of the quantity *?b* directly influences the magnitude of the quantity *?a*. Therefore, one such inference rule states that to achieve the goal of increasing *?a* one could find a quantity *?b* for which *(Q+ ?a ?b)* holds and try to achieve the goal of increasing *?b*.

qualitative predicate definitions

These rules provide qualitative representations for the quantitative predicates employed in generating explanations. For example, the predicate *(dif ?q1 ?q2 ?r)* is used by the system for taking the difference between two quantities (*?q1* and *?q2*) and computing the result (*?r*). One of several rules which form the qualitative predicate definition for the *dif* predicate is shown on the right. It states that the magnitude of the quantity *?q1* directly influences the magnitude of the result *?r* in a *dif* predicate. These definitions and the general qualitative inference rules described above are similar to elements of Forbus' Qualitative Process Theory [Forbus84].

```
(rule   :form
        (Q+ ?r ?q1)
        :ants
        (qrelation (dif ?q1 ?q2 ?r))
```

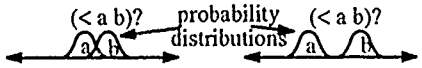
approximation definition rules

Approximate quantities have properties which can be expressed in a qualitative manner. These will be discussed in more detail below.

qualitative probability rules

These rules relate the probabilities of success of predicates in a way similar to the general qualitative inference rules. Proportionalities can be declared between the probabilities of success of certain pairs of predicates as well as between the probability of success of a predicate and the magnitude of a quantity. Using these proportionalities, goals of achieving increases or decreases in probabilities of success can be achieved. For example, the probability of success of the antecedent to a rule is declared to have a positive influence on the probability of success of the consequent of a rule.

In order to see how the qualitative tuning explanation is constructed using these rules, it is important to understand how qualitative probabilities of success are related to tunable quantities. Quantitative predicates employed by the system have one of two basic intents. Either they are *calculation predicates*, whose purpose is to compute some value (e.g. the *dif* function discussed earlier), or they are *test predicates*, which are designed to fail for certain sets of inputs (e.g. the *less-than* function). There is no way to vary the probability of success of a calculation predicate since they always succeed. A test predicate's probability of success, is sensitive to the probability distribution of its argument quantities. In the diagram below, the less-than test on the



right has a higher probability of succeeding given the illustrated probability distributions for its arguments. While probability distributions are difficult to define and work with, there is a much simpler qualitative view of the probability distribution: probability density decreases as one

moves either higher or lower away from the central value. The general definition for a data approximation embodies this principle. The measured quantity is taken to be the central value. Some distribution is present because of the uncertainty involved. Without knowing any details about the distribution, the definition for a data approximation states that the probability of encountering the actual value for the measured quantity decreases as we get farther from the measured approximate value. One of the approximation definition rules regarding data approximate quantities is shown below

```
(rule :form
      (PQ- (< ?test ?loc) ?test)
      :ants
      (data-approx-quantity ?loc2)
      (IQ+ ?loc ?loc2))
```

This translates to: *if a less-than is being performed between ?test and a quantity ?loc which is indirectly or directly qualitatively proportional to a data approximate quantity, the probability of the less-than succeeding is inversely proportional to the magnitude of the ?test quantity.*

Rules like this effectively translate goals to increase the probability of success of a predicate into goals to increase or decrease quantities.

In general, an explanation for positively influencing the probability of a predicate proceeds so as to:

1) relate the probability of the failing predicate to that of a test predicate involving approximate quantities
2) use the definition of a data approximation to relate the probability of success of a test predicate with the magnitude of a tunable quantity

To guarantee that the probability of the failing predicate will increase, all the test predicates in the rule antecedents must be examined. At least one must show an increasing probability of success and the others must be non-decreasing.

The tuning explanation, once generated, indicates only which quantities to tune and in which direction. The remaining task is to carry out the tuning of the rule approximations. Figure 3 illustrates three possible states during the tuning of a rule approximate quantity. Bounds and increasing or decreasing constraints all have general predicates associated with them which allow a specific numeric value for the bound to be derived in the current context. The ordering of the constraints is required to be the same across multiple contexts in which the rule approximation is used. Values to the left of a lower bound or to the right of an upper bound are not supported by explanations in the approximate model

of the world. Tuning takes place between these bounds through the posting of increasing and decreasing constraints. When new constraints are posted to a rule approximation a function called the *quality function* is re–calculated. The quality function provides an estimate of which values and ranges of values best satisfy the current constraints of the rule approximation. It is scaled to between –1 and 1 where a negative value means the current set of constraints is not met and a positive value indicates they are. The magnitude indicates a relative rating of how well or how poorly the constraints are met. For example, for a rule approximation with bounds but no increasing or decreasing constraints, the function is flat between the bounds indicating there is no preference in choosing values in that region.

The different rule approximation types described earlier use the quality function in different ways. A constraint approximation uses the quality function to give a rating for how well a value meets the constraints in the current context. A control approximation, on the other hand, uses the quality function in generating a set of approximate rules which control the choice for its controllable quantity. In generating the approximate rule set of a control approximation, preference is given to values closer to the current value of the controllable quantity. For the unconstrained case of Figure 3 three rules are generated: one to prefer the lower bound when the current value is less than the lower bound, one to keep the current setting if it lies between the bounds, and one to use the upper bound if the current value is greater than the upper bound. When an expense is associated with movement of a control, this minimizes the expenditure. In all other cases, for control approximations, the rule sets the

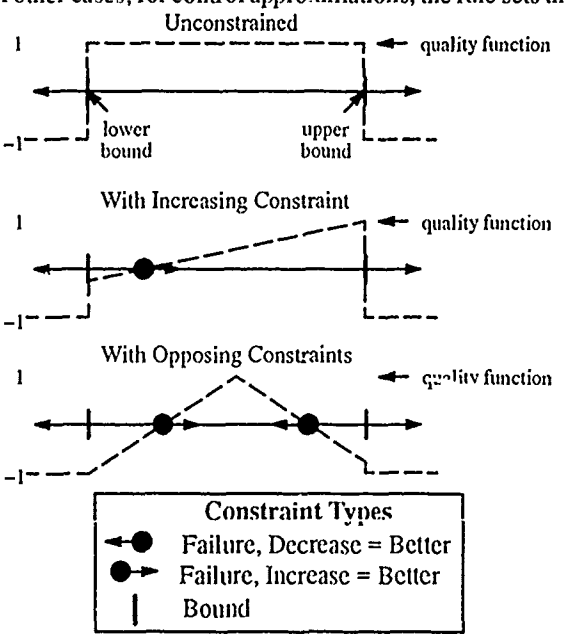


Figure 3. Three Possible States for Constraints on a Rule Approximate Quantity

control to correspond with the peak value of the quality function.

Therefore, to carry out the tuning as prescribed by the qualitative tuning explanation, a new constraint is posted to the appropriate rule approximation for each suggested tuning. If the value at which the failure was suggested was originally generated from one of the constraints or bounds of the approximation, the same general predicate expression is used for calculating it but the type of constraint is changed as necessary. When constraints need to be posted between sets of existing constraints, a new general expression is created using the general expressions for the two surrounding constraints and using the ratio between their specific values in the context of the failure. Once the new constraint has been added (or the old constraint changed) the quality function is re–computed and for control approximations, the corresponding approximate rules replaced by a new set which reflects the new quality function.

With constraint approximations, another decision also must be made before tuning. When the qualitative tuning explanation indicates that the tunable quantity related to a constraint approximation is the target for tuning, it is possible that the current constraint approximation had no say in the choice that failed. This is because constraint approximations are combined using approximate weights. If the quality function of the current constraint approximation did give the selected value a negative rating, the associated weight approximation should be tuned instead of the constraint itself. This serves to increase the relative importance of a constraint which is already tuned correctly. Since weights are scaled in the range 0 to 1, this amounts to either tuning the indicated weight to be increased from the current value or equivalently tuning all the weights for the other constraints employed in the rating function to be decreased from their current values if the indicated weight is already set to 1.

4. An Example Illustrating the Algorithm

Our testbed for learning and tuning approximate explanation–based rules is the robotic grasping domain. The system, GRASPER, is implemented in Lucid Common Lisp running on an IBM RT125. The system acquires visual object contour data using an MV1 frame grabber connected to an overhead mounted CCD camera (see Figure 4). The system controls a UMI RTX scara–type robotic manipulator equipped with variable force control and joint encoders for all joints.

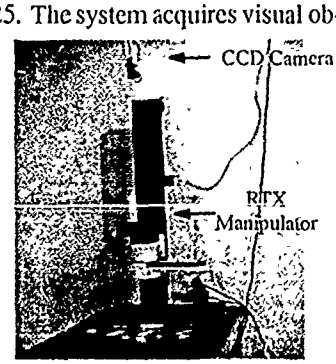


Figure 4. RTX Setup

Initially, the system uses the camera to acquire contour information about objects in the workspace. These contours are then approximated with n-gons (internal data approximations) which result in $(n^2-n)/2$ possible unique grasping face pairs. In runs performed here, n was set to 5. The data approximated object representations as well as the current information about the state of the robot manipulator are asserted in the initial situation. The target object is then selected and an explanation is generated for how to achieve a grasp of the target. Figure 5 (automatically drawn by the implementation) shows the selected target object with the visually sensed contour drawn with a heavy line. The light pentagon is the data approximation for the object. The arrows indicate the positions of the leading edges of the fingers for the grasp position given by the produced explanation. The explanation for achieving *grasp-object* involves a total of about 300 nodes with a maximum depth of 10 levels. This explanation was produced using the approximations in their initial state before tuning. The approximate rule employed in the explanation for deciding the gripper opening width to choose once the grasping faces had been selected is shown in Figure 6. This rule affected the separation of the arrows shown in Figure 5. After the explanation was generated, and its associated operator sequence executed, the monitored action shown in Figure 7 encountered a violation of the expected sensor readings.

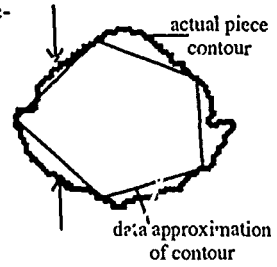


Figure 5. Grasp Target

The original explanation for the *no-gripper-collision-object* goal indicated in the above monitored action is now suspect due to the violated expectations. A sketch of the specific explanation is shown in Figure 8. This explanation for

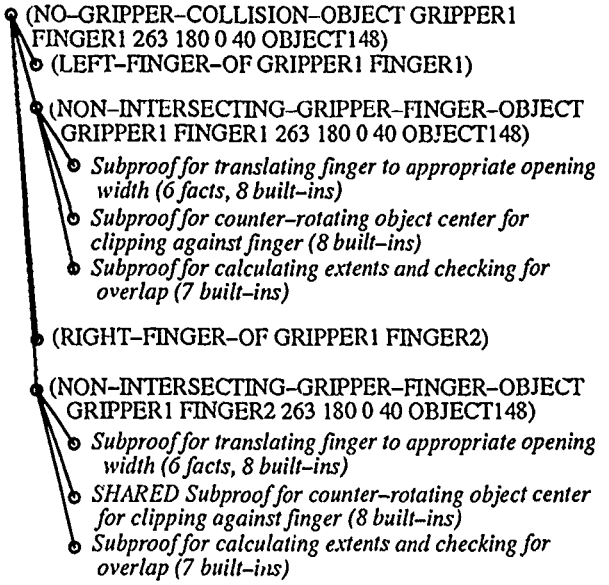


Figure 8. Explanation Specific to Failure

why no external force should have been sensed during the downward move of the gripper is the starting point for developing the qualitative tuning explanation. The generalized consequents and antecedents of the *no-gripper-collision* subproof are asserted as qualitative relations. Approximate quantities employed in the subproof are identified and asserted as such. A proof is then constructed for increasing the probability of success of the *no-gripper-collision-object* goal. Figure 9 shows the qualitative explanation for how opening the gripper (increasing the opening-width tunable quantity) positively influences the

```
INTRA-RULE. R190 ← one of three rules which are initially defined by the opening width rule approximation
   FORM:  (CHOSEN-OPENING-WIDTH ?GRIPPER ?X ?Y ?ANGLE ?OBJECT ?RETURN)
   ANTS:  (GRIPPER-OPENING ?GRIPPER ?LOP187)                          find minimum required
          (GRIPPER-PERP-WIDTH ?GRIPPER ?SPAN)                         opening so fingers don't
          (MIN-SPAN-FOR-OBJECT ?OBJECT ?X ?Y ?ANGLE ?SPAN ?LEFT ?RIGHT) ← collide with object in
          (SUM ?LEFT ?RIGHT ?RETURN)                                    approximate model
          (MAX-GRIPPER-OPENING ?GRIPPER ?MAX-OPEN)
          (<= ?RETURN ?MAX-OPEN) ← can't do it even in approximate model if too wide for gripper
          !
          (< ?LOP187 ?RETURN)
   APPROX: CHOSEN-OPENING-WIDTH ← pointer to parent rule approximation
```

Figure 6. One of the Initial Approximate Rules For Opening Width

```
(MONITOR   (MOVE-ZED ?GRIPPER16601 DOWN 20 64 20 POSITION) ← move down
           (AND  (POSITION ZED ?ZPOS15923) ← force position to be recorded
                 (FORCE ZED ?ZFOR15924)
                 (< ?ZFOR15924 30))  ← all sensed forces on this joint must be less than 30 units
           (POSITION ZED ?LEVEL15925) ← terminate when position is 0 (at the table)
           NIL ?DOC15926
           (NO-GRIPPER-COLLISION-OBJECT ?GRIPPER16601 ?X16235 ← justification for sensor
           ?Y16236 ?ANGLE16237 ?WIDTH16238 ?OBJECT16593))       expectations
                                                  ← variables bound by antecedents to grasp rule
```

Figure 7. The Failing Monitored Action

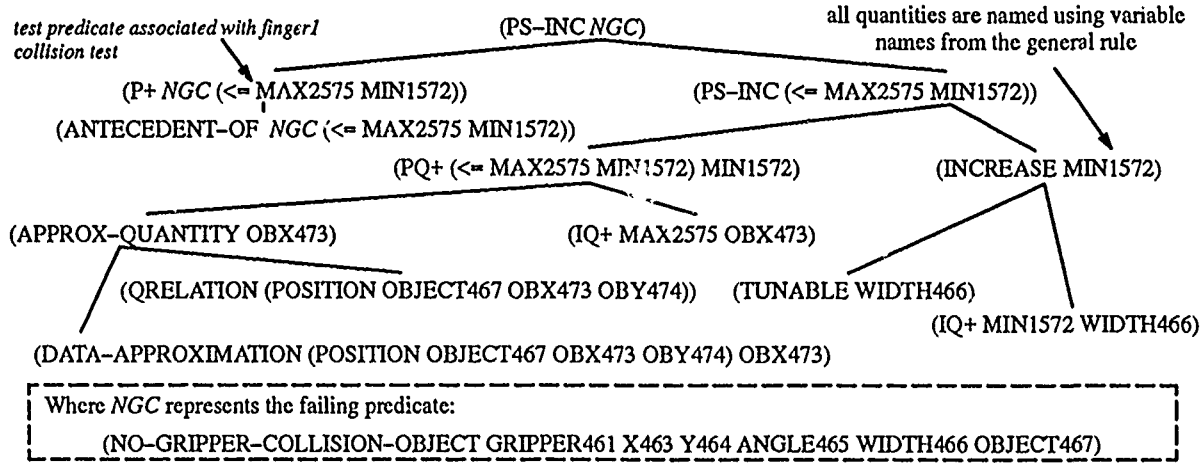


Figure 9. A Qualitative Tuning Explanation

probability that there will be no collision between the first gripper finger and the object. Table 1 gives the semantics

Table 1. Predicates Employed in the Tuning Explanation of Figure 9

(PS-INC ?pred)
the probability of success of ?pred is influenced positively
(P+ ?pred1 ?pred2)
the probability of success of ?pred2 influences the probability of success of ?pred1 positively
(ANTECEDENT-OF ?pred1 ?pred2)
?pred2 is an antecedent of ?pred1 in the rule being analyzed
(PQ+ ?pred ?quant)
the magnitude of the quantity ?quant influences the probability of success of ?pred positively
(INCREASE ?quant)
the magnitude of the quantity ?quant is influenced positively
(APPROX-QUANTITY ?quant)
?qant is an approximate quantity from a data approximation (not controllable by the system)
(IQ+ ?q1 ?q2)
the magnitude of quantity ?q2 indirectly influences the magnitude of quantity ?q1 positively
(TUNABLE ?quant)
the magnitude of quantity ?quant is a tunable quantity from a rule approximation (controllable by the system)

for the predicates employed in the explanation. The topmost left-hand subtree establishes that the probability of the <= test predicate can influence the probability of the *no-gripper-collision* goal because it is an antecedent of the rule. The right-hand subtree establishes that the probability of the <= can be positively influenced through an increase in the opening width. The IQ+ predicate is a built-in predicate for establishing transitive relations between quantities. It consults the body of qualitative proportionalities which hold in the current situation. There is a similar supporting explanation for the other finger, which moves oppositely while opening. Together, the two subproofs cover all the test predicates employed in the original explanation and thus guarantee that opening the gripper wider decreases the chance of this failure (with the target object) happening in the future.

The qualitative tuning explanation indicates that the chosen-opening-width rule approximation should be tuned. Namely, that an increasing constraint be posted at the minimum opening width, which was chosen in the failure. Figure 10 illustrates the chosen-opening-width rule approximation before (top) and after (bottom) tuning has occurred. After tuning, the rules associated with the rule approximation are redefined. Afterwards, the single new rule associated with this approximation reads:

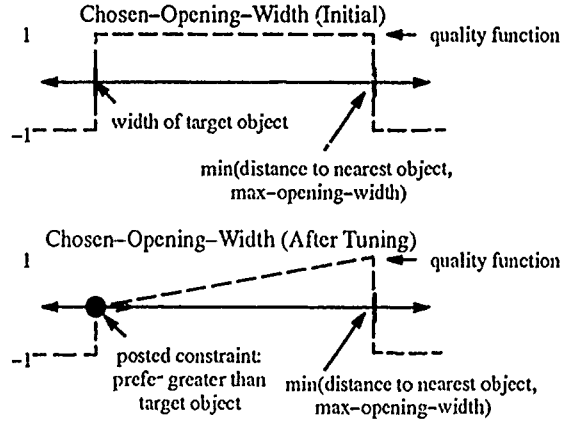


Figure 10. The Chosen-Opening-Width Rule Approximation Before and After Tuning

```
INTRA-RULE: R30278
    FORM:
        (CHOSEN-OPENING-WIDTH ?GRIPPER ?X ?Y
         ?ANGLE ?OBJECT ?RETURN)
    ANTS:
        (GRIPPER-PERP-WIDTH ?GRIPPER ?SPAN)
        (DISTANCE-TO-CLOSEST-OBJECT ?OBJECT ?X ?Y
         ?ANGLE ?SPAN ?RADIUS)
        (GRIPPER-FINGER-PARALLEL-WIDTH ?GRIPPER
         ?PSPAN)
        (DIF ?RADIUS ?PSPAN ?NRADIUS)
        (MAX-GRIPPER-OPENING ?GRIPPER ?MAX-OPEN)
        (MIN ?NRADIUS ?MAX-OPEN ?RETURN)
        (MIN-SPAN-FOR-OBJECT ?OBJECT ?X ?Y ?ANGLE
         ?SPAN ?LEFT ?RIGHT)
        (SUM ?LEFT ?RIGHT ?MIN)
        (<= ?MIN ?RETURN)
    APPROX: CHOSEN-OPENING-WIDTH
```

This rule prefers selection of the peak of the newly re-calculated quality function which corresponds to opening as wide as the current situation permits.

5. Experimental Results

The GRASPER system was given the task of achieving equilibrium grasps on the 12 smooth plastic pieces of a children's puzzle. Figure 12 shows the gripper and several of the pieces employed in these experiments. A random ordering and set of orientations was selected for presentation of the pieces. Target pieces were also placed in isolation from other objects. That is, the workspace never had pieces near enough to the grasp target to impinge on the decision made for grasping the target. The first run was performed with approximation tuning turned off. The results are illustrated in Figure 11. Failures observed during this run included *finger stubbing failures* where a gripper finger struck the top of the object while moving down to surround it and *lateral slipping failures* where, as the grippers were closed, the object slipped out of grasp, sliding along the table surface. In the system's initial approximate representation for the world, the choice of grasping faces is constrained only by the gripper being able to open wide enough to surround them and that an equilibrium grasp is realizable with the current gripper-object friction coefficient (initially 1 here). Since a friction coefficient of 1 is likely to be high for these materials, the choice of contact faces is likely to be under-constrained initially, resulting in slipping failures. The choice of opening width is the minimum deviation from the current opening width (initially 0 here) which satisfies the approximate model of the grasp. Due to uncertainties in the world, this approximate opening width may often result in stubbing failures. Therefore, the error rate of our initial approximate plan was high resulting in 9 finger stubbing failures and 1 lateral slipping failure in 12 trials.

Figure 12. Gripper and Pieces

In our second run, approximation tuning was turned on. An initial stubbing failure on trial 1 led to a tuning of the chosen-opening-width rule approximation which determines how far to open for the selected grasping faces. Since the generated qualitative tuning explanation illustrated that opening wider would decrease the chance of this type of a failure, the system tuned the approximate rule to choose the largest opening width possible (constrained only by the maximum gripper opening and possible collisions with nearby objects). In the case

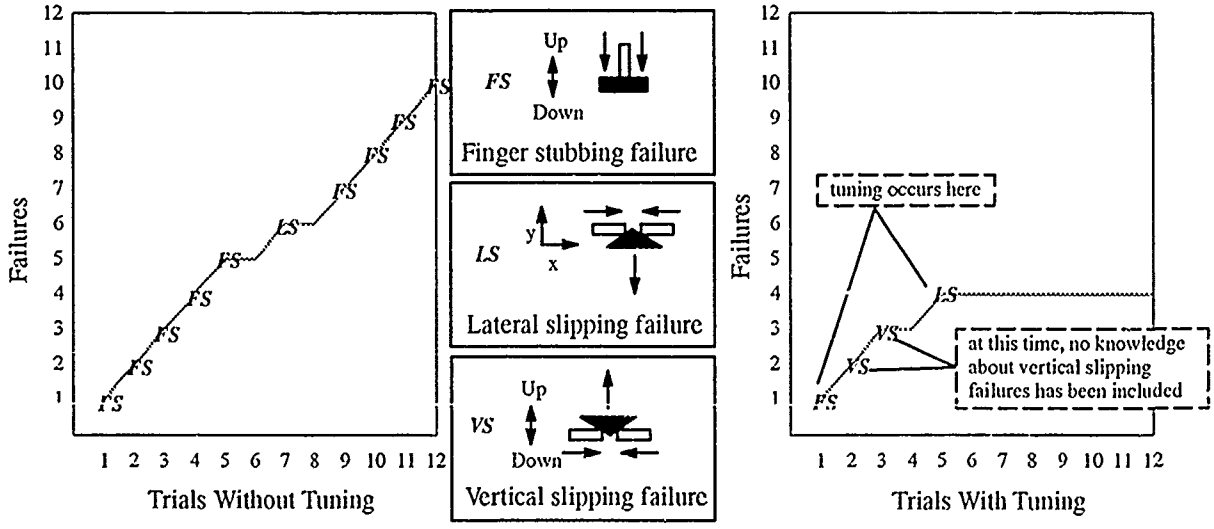


Figure 11. Comparison of Tuning to Non-tuning in Grasping the Pieces of a Puzzle

of isolated grasp targets, opening to the maximum gripper width is preferred. In trials 2 and 3, finger stub failures didn't occur as they had previously because the opening width was greater than the object width for that orientation. *Vertical slipping failures*, which the current implementation does not currently have knowledge about, did occur. The system had to be told that a vertical slipping failure had occurred instead of the lateral slipping failure it thought had occurred. This is because, without further knowledge about vertical slipping failures and a means for detecting them, they look in other ways (the force vs. position profile of the gripper closing) like a lateral slipping failure. Preventing vertical slipping failures involves knowing shape information along the height dimension of the object, which we plan to give in the future using a model-based vision approach. In trial 5, a lateral slipping failure is seen and the qualitative tuning explanation suggests decreasing the contact angle between selected grasping surface through tuning the *contact-angle-constraint* approximation. This is tuned to prefer smaller contact angles. A single tuning for the finger stubbing and lateral slipping failures was sufficient to eliminate those failures with isolated grasp targets.

6. Future Work & Conclusions

Further experiments are in progress with the system at this time. One involves investigating approximation tuning in environments where the target object is nearby to other objects. This type of situation will illustrate tradeoffs between constraints on an approximation. For instance, because of the close proximity of other objects, failures may occur because opening to the maximum width (as was learned here) results in a collision with a nearby object. We hope to demonstrate that the approximations tune themselves appropriately for different characteristics of the environment, such as density of objects.

We are also in the process of implementing a general mechanism for rating the plausibility of different possible failures based on the qualitative view of an explanation discussed here. By isolating the cause of the failure more precisely within the failed expectation explanation, through the plausibility rating mechanism, qualitative tuning explanations are made easier to generate.

Any system which hopes to manage a model of the world in conjunction with actions and values sensed in the real world has to have some approximation mechanism. The characterization of data and rule approximations provides a good framework from which to explore how to manage approximations. Tunable approximate quantities are sufficiently powerful to introduce uncertainty tolerance into plans in an on-demand manner. Uncertainty tolerance allows plans to function in spite of the type of data errors which leads to failures. The approximation tuning method presented offers a general way of discovering the relationships between the tunable approximate quantities, data approximate quantities measured from the world, and ultimately the probability of success of a goal. The experimental results indicate the method of approximation tuning decreases the overall failure rates for real-world domains as compared with a static approach which must continue to learn and explain within its fixed declared representation of the world.

Acknowledgements

I would like to thank my advisor, Gerald DeJong, for his many helpful discussions and insightful ideas as well as comments on the draft of this paper. Jonathan Gratch was also helpful in comments on the new classes of approximations introduced here. This research was supported by the Office of Naval Research under grant ONR N00014-86-K-0309.

References

[Bennett89a] S. W. Bennett, "Learning Uncertainty Tolerant Plans Through Approximation in Complex Domains," M.S. Thesis, ECE, University of Illinois, Urbana, IL, January 1989. (Also appears as Technical Report UILU-ENG-89-2204, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign)

[Bennett89b] S. W. Bennett, "Learning Approximate Plans for Use in the Real World," *Proceedings of the Sixth International Conference on Machine Learning*, Ithaca, NY, June 1989, pp. 224-228.

[Chien89] S. A. Chien, "Using and Refining Simplifications. Explanation-based Learning of Plans in Intractable Domains," *Proceedings of The Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 590-595.

[DeJong86] G. F. DeJong and R. J. Mooney, "Explanation-Based Learning. An Alternative View," *Machine Learning 1*, 2 (April 1986), pp. 145-176.

[Doyle86] R. J Doyle, "Constructing and Refining Causal Explanations from an Inconsistent Domain Theory," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 538-544.

[Forbus84] K. D. Forbus, "Qualitative Process Theory," *Artificial Intelligence 24*, (1984), pp. 85-168.

[Hammond86] K. Hammond, "Case-based Planning: An integrated theory of planning, learning, and memory," Research report no 488, Yale University, Dept. of Computer Science, New Haven, CT, 1986.

[Keller87] R. M. Keller, "The Role of Explicit Contextual Knowledge in Learning Concepts to Improve Performance," Ph.D. Thesis, Department of Computer Science, Rutgers University, New Brunswick, NJ, January 1987.

[Mitchell86] T. M. Mitchell, R. Keller and S. Kedar-Cabelli, "Explanation-Based Generalization: A Unifying View," *Machine Learning 1*, 1 (January 1986), pp. 47-80.

[Mooney86] R. J. Mooney and S. W. Bennett, "A Domain Independent Explanation-Based Generalizer," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 551-555.

[Schank82] R. C. Schank, *Dynamic Memory*, Cambridge University Press, Cambridge, England, 1982

[Segre87] A. M. Segre, "Explanation-Based Learning of Generalized Robot Assembly Tasks," Ph.D. Thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL, January 1987.

[Zweben88] M. Zweben, "Improving Operationality with Approximate Heuristics," *Proceedings of the American Association for Artificial Intelligence Spring Symposium on Explanation-Based Learning*, Palo Alto, CA, March 1988, pp. 100-106.

Correcting and Extending Domain Knowledge using Outside Guidance*

John E. Laird and Michael Hucka and Eric S. Yager and Christopher M. Tuck
Artificial Intelligence Laboratory
The University of Michigan
1101 Beal Ave.
Ann Arbor, MI 48109-2110

Abstract

Analytic learning techniques, such as explanation-based learning (EBL), can be powerful methods for acquiring knowledge about a domain where there is a pre-existing theory of the domain. One application of EBL has been to learning apprentice systems where the solution to a problem generated by a human is used as input to the learning process. The learning system analyzes the example and is then able to solve similar problems without outside assistance. One limitation of EBL is that the domain theory must be complete and correct. In this paper we present a general technique for learning from outside guidance that can correct and extend a domain theory. In contrast to hybrid systems that use both analytic and empirical techniques, our approach is completely analytic, using the chunking learning mechanism in the Soar architecture. This technique is demonstrated for a block manipulation task that uses real blocks, a Puma robot arm and a camera vision system.

1 Introduction

Learning through interacting with a human is an efficient method to increase the knowledge of an intelligent agent. Initially, an intelligent agent may have only very general abilities and may require significant guidance from a human operator. Through its experiences, the agent can become more and more autonomous, making most decisions on its own and requiring guidance only for novel situations. It can increase its repertoire of methods for solving problems, improve its reaction time to events in the environment, learn to notice new properties of objects in the environment, as well as refine and extend its own model of the domain.

*This research was sponsored by grant NCC2-517 from NASA Ames and ONR grant N00014-88-K-0554.

Many approaches are possible for learning from experience and outside guidance. In the simplest case, the human solves a problem and the learning system must "watch over the shoulder" of the human as the problem is solved. This is the scheme used in robotic programming systems where a human leads the system through a fixed set of commands to achieve a goal. When the commands are stored, the system can perform only that one task and there is no conditionality in the learned plan. The robot will execute exactly the learned set of actions, independent of the state of the environment.

To avoid these problems, "learning apprentices" have been developed that create generalized plans, indexed by the appropriate goal. These systems, such as LEAP [Mitchell *et al.*, 1985] are based on a learning strategy called explanation-based learning (EBL) [DeJong & Mooney, 1986; Mitchell *et al.*, 1986]. In a learning apprentice system, the human provides a complete solution to a problem. An underlying "domain theory" is then used to "explain" the actions of the human expert. From the derived explanation, all of the dependencies between the actions are recovered and a general plan is created.

Two extensions to this basic model of external guidance have been previously made using the Soar architecture which uses an analytic learning mechanism similar to EBL, called *chunking* [Golding *et al.*, 1987; Laird *et al.*, 1987; Rosenbloom & Laird, 1986]:

1. The learning through guidance is integrated with general problem solving and autonomous learning. The system can learn from its own experiences with or without outside guidance.
2. The system actively seeks advice while solving its own problems as opposed to passively monitoring a human problem solver. In addition, the guidance occurs in the context of the system solving a problem and the guidance is at the level of individual decisions instead of complete plans.

In this paper we will demonstrate that this technique can be extended to learning new domain knowledge, not just control knowledge. The tasks we will use

involve simple manipulations of blocks using a Puma robot arm and a single camera machine vision system. This a simplification of the manipulator control tasks performed by Segre's ARMS system [Segre, 1987]; however, ARMS worked only in a simulated environment. With the real robot, the domain theory is complicated by incomplete and time-dependent perception; the camera provides only two-dimensional information and is sometimes obscured by the robot arm, and the vision processing time is 4 seconds. The goal of the task is to line up a set of blocks that have been scattered over the work area. In one task, all of the blocks are simple cubes that the gripper can pick up in two different orientations. In the second task, one of the blocks is a triangular prism. The gripper is unable to pick up the prism when it closes over the inclined sides. Instead it must be oriented so that it closes over the vertical faces of the block.

One restriction on using analytic methods such as EBL and chunking is that they require a complete and correct domain theory. The domain theory is an internal model of all of the preconditions and effects of the operators used to perform the tasks. For the block manipulation task, the operators are commands to the robot controller such as open gripper, close gripper, and withdraw gripper from work area. If the internal model of these operators is in some way incorrect, then the control knowledge learned through guidance will also be incorrect. For example, if the domain knowledge is not sensitive to the special properties of a prism block, the control knowledge it learns will ignore the orientation of a prism block when attempting to pick it up. We will show how it is possible to correct control knowledge using outside guidance and a domain theory for determining the relevant features of the environment and relating them to robot actions.

Even if the learned control knowledge can be corrected, the original domain theory may still be incorrect. We show that our system can correct the original domain theory by replacing incorrect operator definitions with new, corrected definitions, and that it can extend the domain theory by creating completely new operators. In both cases, no modification is made to the underlying architecture; instead knowledge is added to correct and create operators. In the block manipulation task, we will demonstrate the acquisition of planning and execution knowledge for the `rotate-gripper` operator. These extensions are based on the creation of an underlying domain theory for the construction of new operators. In the limit, a complete domain theory can be acquire through ouside guidance, with only very limited predefined knowledge of the task.

Section 2 presents the system architecture. Section 3 gives an example of learning control knowledge through outside guidance. This reproduces the results of Golding, Rosenbloom and Laird (1987) but in a task requiring interaction with a real environment. In Section 4 we extend this work by demonstrating the correction of control knowledge using outside guidance.[1] Sections 5 and 6 present demonstrations of the correction and extension of domain knowledge through guidance. The final section discusses the contributions and limitations of the current approach.

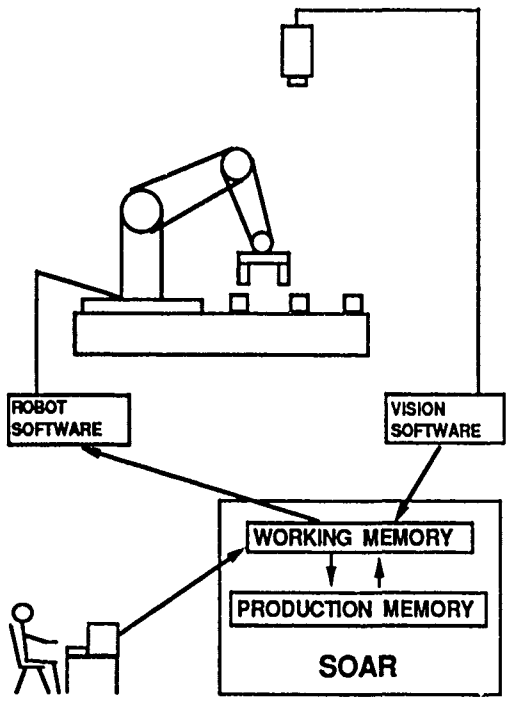


Figure 1: Robo-Soar system architecture.

2 System Architecture

The system we are developing is called Robo-Soar [Laird *et al.*, 1989].[2] Figure 1 shows its architecture. Visual input comes from a camera mounted directly above the work area. A separate computer processes the images, providing asynchronous input to Soar. The vision processing extracts the positions and orientations of the blocks in the work area as well as distinctive features of the blocks. The vision and robotic systems are sufficiently accurate so that there are no significant sensor or control errors for the block manipulation tasks.

In Soar, a task is solved by selecting and applying operators to transform an initial state to some desired state. Operators are grouped into *problem spaces* and Soar selects an appropriate problem space for each goal. For the block manipulation task, the states are different configurations of the blocks and gripper in the

[1] Some of the material in Sections 2, 3 and 4 has been previously presented in Laird *et al.*, (1989).

[2] Robo-Soar is implemented in Soar 5, a new version which supports interaction with external environments [Laird *et al.*, 1990].

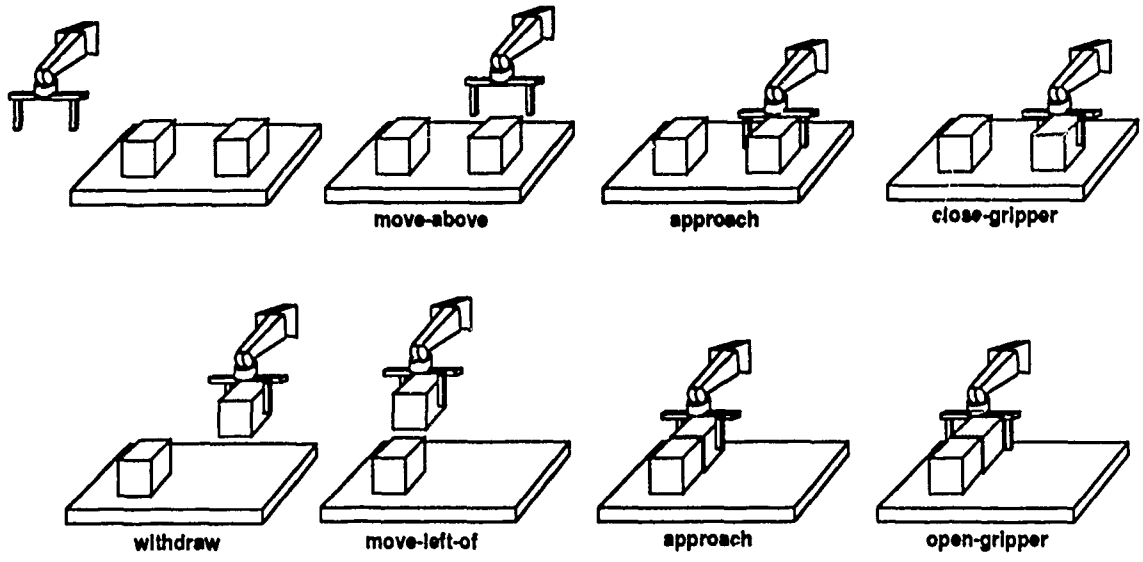


Figure 2: Moving a block using the primitive Robo-Soar operators.

external environment. Some basic operators are shown in the trace of Robo-Soar solving a simple block manipulation problem in Figure 2. These operators generate motor commands for the Puma.

One complication is that the camera is mounted directly above the work area so that the arm obscures the view of a block that is being picked up. Two operators, **snap-in** and **snap-out**, move the arm in and out of the work area so that a clear image can be obtained. These operators are necessary, but for simplicity, they will not be included in any of the examples.

This characterization of Robo-Soar does not distinguish it from any other robot controller. What is different is the way Soar makes the decisions to select an operator. Many AI or robotic systems create a plan that the robot must execute to select one operator after another. Instead of creating a plan, Soar makes each decision based on a consideration of its long-term knowledge, its perception of the environment, and its own internal state and goals. Soar's long-term knowledge is represented as productions that are continually matched against the working memory which includes all input from sensors, guidance from an advisor, and internal data structures and goals. Commands to the robot controller are issued by creating data structures in working memory.

In contrast to traditional production systems such as OPS5, Soar fires all successfully matched production instantiations in parallel, which in turn elaborate the current situation, or create *preferences* for the next operator to be taken. Soar's language of preferences allows productions to control the selection of operators by asserting that operators are acceptable, not acceptable, better than other operators, as good as others, and so on. Production firing continues until no additional productions match, at which point, Soar examines the preferences and selects the best operator for the current situation. Once an operator is selected, productions can fire to apply the operator. If the operator affects the external environment, the productions create commands to the motor system. If the operator is used for planning or internal processing, the productions directly modify the internal data structures in working memory. Additional productions test for operator completion and signal that a new operator can be selected.

In a familiar domain, Soar's knowledge may be adequate to select and apply an operator without difficulty. However, when the preferences do not determine a unique choice, or when the productions are unable to implement the selected operator, an *impasse* arises and Soar automatically generates a subgoal. In the subgoal, Soar uses the same approach; it selects and applies operators from an appropriate problem space to achieve the subgoal. The operators in the subgoal can modify or query the environment, or they may be completely internal, possibly simulating external operator applications on internal representations.

When Soar creates results in its subgoals, it learns productions, called *chunks*, that summarize the processing that occurred. The actions of a chunk are based on the results of the subgoal. The conditions are based on only those working-memory elements that were tested and found *necessary* to derive the results. Thus, knowledge used to control the selection of operators in the subgoal is not included in the derivation because it affects only the efficiency of producing the results, not their correctness.

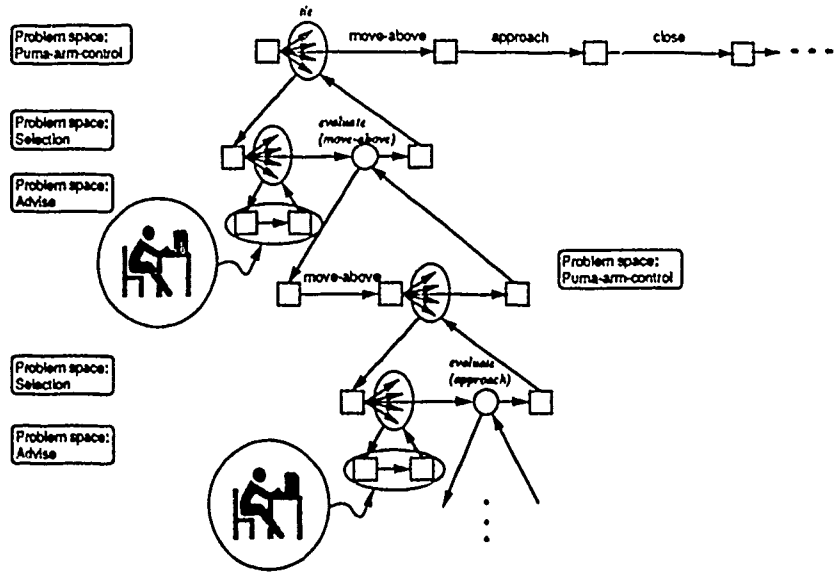


Figure 3: Trace of problem solving using external guidance to suggest appropriate task operators for evaluation. The problem solving goes left to right with squares representing states, while horizontal arcs represent operator applications. Downward pointing arcs are used to represent the creation of subgoals, and upward pointing arcs represent the termination of subgoals.

3 Learning Control Knowledge

To attempt the block manipulation task, knowledge about the operators must be encoded as productions. This knowledge includes productions that suggest operators whenever they can be applied legally, as well as productions that implement the operators by creating motor commands to move the arm. Because the feedback from the vision system is incomplete when the arm is being used to pick up a block, some productions must also create internal expectations of the position of blocks until feedback is received when the arm is moved out of the way.

With just this basic knowledge, Robo-Soar can attempt a task, but it will encounter impasses whenever it tries to select a task operator, as shown in Figure 3. To resolve these *tie* impasses, the tied task operators are evaluated in a subgoal and preferences are created to pick the best one. The evaluations are carried out by operators created in the subgoal of the tie impasse. Within this subgoal, a decision must be made as to which evaluation operator should be selected first, and thus, which task operator should be evaluated first. If the task operator that leads to the goal is evaluated first, the other task operators can be ignored because the 'best' operator has been found. As with the original decision to select between the task operators, the decision to select an operator to be evaluated will lead to a tie impasse.

Outside guidance can be used to select the best operator to evaluate. The guidance is used to determine which operator to evaluate first, not which operator to apply. The advantage of this approach is that the guidance acts only as a heuristic that is verified by internal problem solving. The problem solving calculates the internal evaluation and determines whether the chosen operator is really appropriate. The system can then learn the conditions under which the operator is appropriate. If advice directly selected a task operator that led to motor commands, there would be no internal analysis of the operator that could be used for learning.

To acquire the guidance within the subgoal, Robo-Soar uses its *advise* problem space which has operators that print out the acceptable task operators, ask for guidance, and wait for a response. If guidance is available, the appropriate operator is selected for evaluation. If no guidance arrives while the system is waiting, a random selection is made. All guidance in Soar takes this form where the advisor selects between competing operators. This restricts the guidance to be from predefined alternatives (the tied operators) and does not allow the input of arbitrary data structures.

Once an operator has been selected to be evaluated, an impasse arises because there are no productions that can directly compute the evaluation. The default response to this impasse is to simulate the task operator on an internal copy of the external environment. This requires an internal model of the preconditions and effects of the operators which correspond to the domain theory of an EBL system. The internal search continues through the recursive creation of tie-impasses, advice, and evaluation until a state is found that achieves the goal.

After the goal is achieved within the internal search, preferences are created to select those operators evaluated on the path to the goal. Each of these preferences is a result of a tie-impasse, and chunks are built to summarize the processing that led to their creation. This processing includes all the dependencies between the operator's actions and the preconditions and actions of the operators that were applied after it to finally achieve the goal, essentially the same as the goal regression techniques in EBL [Rosenbloom & Laird, 1986]. Figure 4 is an example of the production that is learned for the **approach** operator. Notice that the production not only tests aspects of the current situation, but also aspects of the goal. The productions learned from this search are quite general and do not include any tests of the exact positions or names of the blocks because these features were not tested in the subgoal. The internal search is based on relationships such as **left-of** or **above** instead of the exact x, y, and z locations of the blocks.

```
If the approach operator is applicable, and
   the gripper is holding nothing
      in the safe plane above a block,
   and that block must be moved
     to achieve the goal,
then create a best preference for
     the approach operator.
```

Figure 4: Example production learned by Robo-Soar.

Following the look-ahead search, Robo-Soar has learned which operator to apply at each decision point. Robo-Soar applies the operators by generating motor commands and moving the real robot arm. This is not, however, a blind application of a plan. Each of the learned productions will test aspects of the environment to insure that an operator is selected only when appropriate.

4 Correcting Control Knowledge

A problem with the analytic learning approaches such as EBL and chunking is that the learning is only as good as the underlying knowledge. If there is an error in the original domain theory, the learning will preserve the error. External guidance is of no help when restricted to suggesting task operators because the error can be in the underlying implementation of operators used by the learning procedure.

We consider a simple case of this problem by attempting the same task as before except with blocks shaped as triangular prisms. If the original operators were encoded with only cubes in mind, all of the control knowledge and underlying simulation would be insensitive to a feature in the input that must be attended to. To the Robo-Soar vision system, the prisms look just like cubes, except for a line down the middle at the apex of the triangle. In order to pick up these blocks, the gripper must be aligned with the vertical faces of the block, not just any two sides as with a cube. Figure 5 shows the operators Robo-Soar applies for this problem. If the gripper is not correctly aligned, the gripper will close but not grasp the block. Upon withdrawing the gripper, the block will not be picked up.

There are many possible machine learning approaches that could be used to correct the underlying knowledge. First, the system could have an underlying "subdomain" theory [Doyle, 1986] of inclined planes, grippers, and friction that it uses to understand why the block was not picked up. This requires knowing beforehand that this knowledge will be necessary, and for many tasks this additional domain knowledge may be difficult to obtain. A second approach is to gather examples of failure and use inductive learning techniques to hypothesize which feature in the environment was responsible for the problem. This may identify the feature, but it requires many failures and also gives no hint as to the appropriate action. A third approach is for the system to experiment with its available operators to see what actually works [Carbonell & Gil, 1987]. This approach can be quite effective, but it also can be quite time consuming and possibly dangerous. Our approach involves increasing the interaction between the advisor and the robot so that the advisor can point out relevant features in the environment and associate them with the potential success or failure of a given operator or set of operators.

This approach incorporates outside guidance with prior work in Soar on recovery from incorrect knowledge [Laird, 1988]. Instead of correcting the productions, our recovery scheme learns new productions whose actions correct the decision affected by the incorrect production. The advisor provides guidance in re-evaluating the operators being consider for a decision, leading to new productions that correct the error. This process of recovery is a domain-independent strategy encoded as productions.

The recovery method is invoked when the system notices that an error has been made. In Robo-Soar, the vision system detects that the prism block is still on the table following an attempt to pick it up. The decision that must be corrected is the choice of the **approach** operator when, in fact, the gripper should be rotated so that it is aligned with the vertical sides of the prism block. Figure 6 shows an abbreviated trace of the problem solving to correct this decision.

Once an error has been detected, the system tries to push forward to the goal, but more deliberately, so that the errant control knowledge does not select the wrong operator. Previously learned control knowledge is overridden by forcing impasses for every task operator decision. Within the context of each of these impasses, all of the available operators can be evaluated and new preferences can be created to modify a decision if it is incorrect.

The underlying internal domain knowledge that gen-

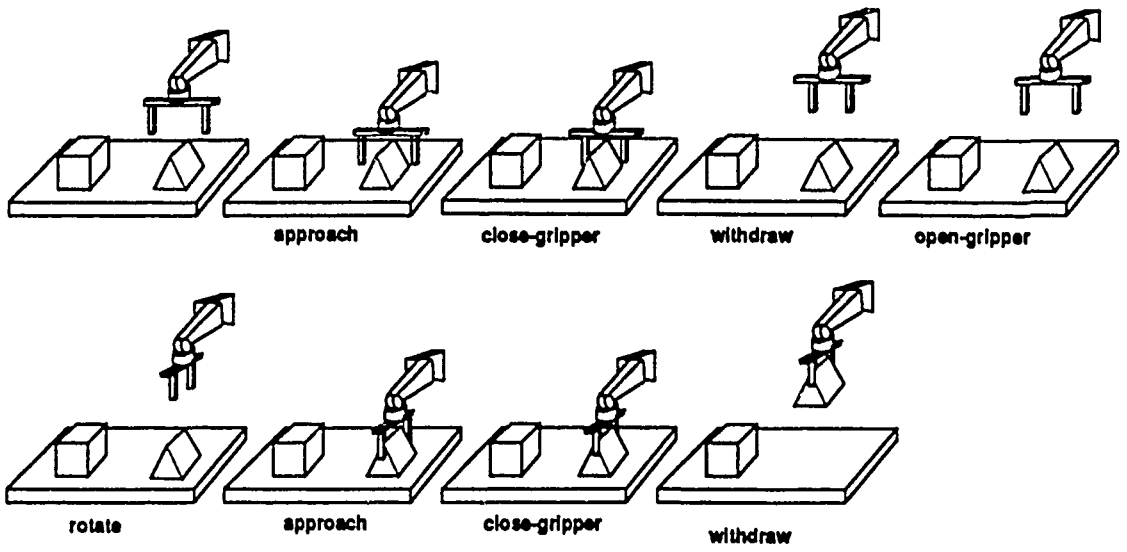


Figure 5: Trace of operator sequence using recovery.

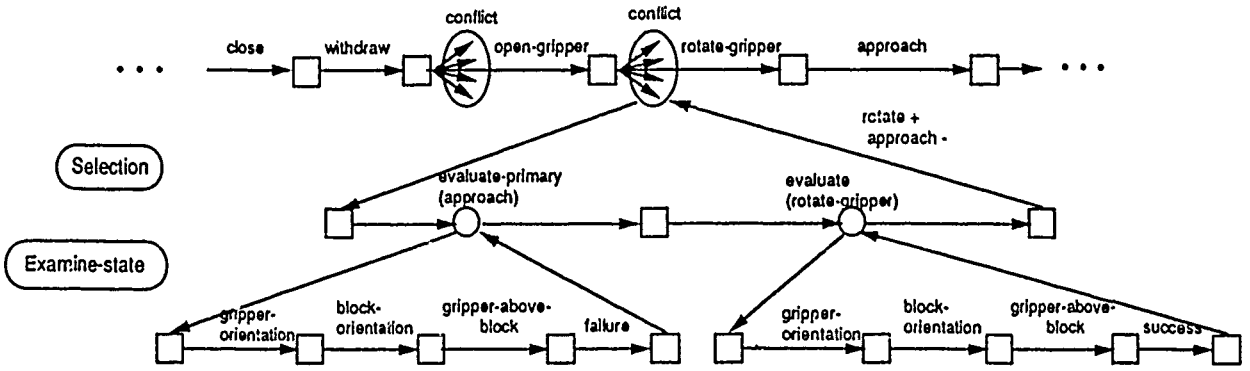


Figure 6: Trace of problem solving with recovery. The subgoals that allow outside guidance are omitted and would be used to select operators in both the selection and examine-state problem spaces.

erates the evaluation may also be incorrect. Therefore, the evaluation process is modified so that outside guidance can be used to evaluate an operator and associate that evaluation with relevant features of the environment. This modified evaluation is performed in the *examine-state* problem space.

The examine-state problem space is an underlying theory for determining the features of the environment that are relevant to the operator being evaluated and relating them to a specific evaluation. There are three classes of operators in the problem space. The first class, called `notice-feature`, explicitly tests the existence of a feature of the current task state or goal. This operator allows this system to explicitly search through the feature space of the task. In our example, this allows the system to test the line down the middle of the prism block, which was previously ignored by the task productions. For future reference in this paper we will call this feature *block-orientation*. The second class of operators, called `compare-features`, can perform simple comparisons between noticed features, such as detecting that two features have the same value. The third class of operators creates an evaluation, such as success or failure, for the task operator being evaluated. Together, these three classes of operators provide a complete domain theory for computing evaluations and relating these evaluations to relevant features of the environment. Although complete, it is underconstrained because any evaluation can be paired with any set of features.

External guidance can lead Robo-Soar to select the operators that notice only those features relevant to the current task state and associate them with the appropriate evaluation. If an operator is deemed to be on the path to success, a preference will be created to prefer it over an operator that leads to fail-

ure. In our example, the advisor first points out that the **approach** operator will fail when the gripper is above a block where the orientation of the operator is not aligned with the block-orientation. The advisor then suggests evaluating the **rotate-gripper** operator, and points out the relevant features that make its selection desirable. Figure 7 shows the productions that are learned to avoid the **approach** operator and select the **rotate-gripper** operator whenever the gripper is not aligned.

```
If the approach operator is applicable, and
   the gripper is above a block, and
   the gripper's orientation is different
       from a line in the middle of the block,
then create a preference to reject approach.

If the rotate operator is available, and
   the gripper is above a block, and
   the gripper's orientation is different
       from a line in the middle of the block,
then create a best preference for rotate-gripper.
```

Figure 7: Example correction productions learned by Robo-Soar.

Once these evaluations are made, the recovery knowledge detects that the previously preferred operator is now rejected, and therefore assumes that the error has been corrected. The error signal is removed so that future decisions are made without forced impasses. From this point, the chunks apply and take Robo-Soar to the solution, as shown in Figure 5. In future situations, Robo-Soar correctly aligns the gripper before approaching a prism. If errors still exist, the advisor can signal this by merely typing **error** to Robo-Soar. The advisor can also signal that an error has been fixed if Robo-Soar is unable to detect it automatically.

5 Correcting Domain Knowledge

Although the method described in the previous section corrects control knowledge, it does not correct the underlying domain knowledge; specifically, it does not add the precondition for the **approach** operator that the gripper be aligned with the block. Although the new control knowledge prevents the operator from being applied, the missing operator preconditions will lead to errors in learning when the operator is correctly applied. Learning will be incorrect because the missing preconditions will not be incorporated in future chunks that depend upon the application of the **approach** operator.

Instead of attempting to modify the productions that propose and implement the approach operator, Robo-Soar creates a new approach operator that replaces the original. The new operator will have the appropriate preconditions and will always be preferred to the original operator, the original is essentially forgotten. The general approach is to create a new operator, notice additional preconditions in the task state, then learn the implementation of the new operator using the original **approach** operator. Throughout this discussion, all guidance is through the advise problem space as described earlier.

To control the creation of the new operator, the selection problem space is augmented so that when there is an error, one alternative is to evaluate a completely new operator. If the advisor selects this alternative, a new operator is created and evaluated to be better than the original, thus replacing it. When the decision is made to evaluate the new operator, the examine-state problem space is used (along with outside guidance) and appropriate control productions are learned to propose and select this new operator.

At this point, the system does not have the knowledge to apply this operator; therefore, when the new operator is selected, another impasse arises. In response to this impasse, the examine-state problem space is again selected, but it has been augmented with an additional operator that can apply a task operator. To build the correct definition of the new operator, **notice-feature** operators are selected through outside guidance to incorporate the missing preconditions, which for our example are the orientation of the gripper and the block-orientation. Following the determination of the appropriate features, additional guidance can specify a task operator that should be used to implement the new operator in the subgoal. In this case, it would be the original **approach** operator. This operator is applied to the task state within the subgoal and the changes it makes to the state are results of the subgoal. These results lead to the creation of chunks that test for the new operator and its preconditions, and then apply the operator to the state. The new, corrected operator replaces the old operator, thus correcting the underlying domain theory.

6 Extending Domain Knowledge

The method we have described for creating a new operator using an existing operator definition can be extended so that a new operator can be learned from scratch through guidance. This is useful for building a new domain theory, as well as completing an existing domain theory. In Robo-Soar, we consider the situation in which the original programmer decided that it was not necessary to include a **rotate-gripper** operator.

We extend the previous approach by adding domain-independent knowledge that can generate the individual actions of an operator. This knowledge is encoded as additional operators in the examine-state problem space. These operators modify or remove existing structures on the state or new task operator, create new intermediate structures, or terminate the new operator. Once the new operator terminates, chunking creates productions that implement it without the

need for further impasses or guidance.

To teach the system an internal simulation of `rotate-gripper`, the orientation of the gripper and the block are noticed, and then the gripper orientation is modified to be the orientation of the block. The exact values (and representations) of the orientations of the block and the gripper are irrelevant. All that is needed is copy a pointer of the orientation of the block on to the data structure representing the orientation of the gripper. From a single example the system learns a general production for implementing `rotate-gripper`.

These extensions are sufficient for creating operators that modify or remove existing structures or create new intermediate structures. They are insufficient for implementing operators that must create new structures with specific symbols that do not pre-exist in working memory. The problem is that there is no way to generate these symbols so that the can be selected by an advisor. This is the one case where guidance by selecting from a fixed set of alternatives breaks down. Fortunately, the only time that specific symbols are necessary is when issuing commands to the motor system. Therefore, we have included an operator that can generate all of the robot command symbols, such as `move`, `open`, and `rotate`. This is the only domain-dependent knowledge that must be pre-encoded in the system.

All of these extensions expand the examine-state problem space so that it has sufficient symbol manipulation capabilities for creating and implementing task operators through the composition of primitive domain-independent operators. However, outside guidance is necessary to control the composition of features and actions so that only legal operator implementations are generated.

In a previous version of Soar, the system was taught to play Tic Tac-Toe from scratch. The system initially had no notion of two player games, three in-a-row, winning, or losing. The system did have an initial representation of the board, the symbols X and O, and the command to make a move. Through outside guidance, operators were created to pick the side to move next, make a move of the chosen side, wait for the opponent to make a move, and detect winning and losing positions.

7 Discussion

There are two major contributions of this work. First, we have demonstrated that it is possible to extend analytic learning systems so that not only can they learn control knowledge using an existing domain theory, but through outside guidance they can also be used to correct and create new domain knowledge. The examine state problem space is somewhat of a brute-force technique to learn new features and operators. It currently requires an outside agent to lead the system through a search of potentially relevant features and actions. Although it may not be considered the most elegant or complex machine learning technique, it allows the advisor to easily correct and extend the system.

This same approach could be used without an outside agent by having the system engage in experimentation. To experiment, the system can guess at relevant features and associated actions. It may pay attention to irrelevant features, and thus create overgeneral or incorrect chunks. But after many interactions with an environment where there is sufficient feedback, it could gradually learn those correct associations. Many powerful heuristics are available to avoid a blind search through this hypothesis space, such as concentrating on new, unknown features, as well as those features that are modified by the operators under consideration. For example, if the system has discovered that the rotate operator is necessary, it could concentrate its search for features relevant to avoiding approach to those features modified by `rotate-gripper`. This approach could also support hybrid methods that involve both outside guidance and experimentation, where the system experiments when on its own but uses guidance when it is available.

The second major contribution is to demonstrate the generality of guidance in knowledge acquisition. The form of guidance we allow is very restricted in that the advisor must pick from a set of available options. One advantage of this scheme is that the advice is given within the context of a specific problem and advice is asked only for those decisions for which the system has incomplete knowledge. A second advantage is that the advisor does not have to make explicit the reasons for the selection of an operator for which the system has a correct internal model; the learning mechanism performs the necessary analysis. A third advantage is that the guidance and learning occur while the system is running. There is no need to ever turn off the performance system to update or correct its knowledge base. Finally, by integrating the guidance, the problem solving and learning within a single architecture such as Soar, the guidance can be used to correct or extend any of the long term knowledge of the system.

The weakness of this approach is that the advisor sometimes must specify individual preconditions and effects of an operator. This can be quite tedious and it requires the human to identify which preconditions or effects that are missing when correcting a domain theory. These problems would be greatly reduced if our interface were improved so that the advisor could more directly observe the structure of the current state and operator. A more long-term solution is to provide the system with additional knowledge that allows it to perform more of the diagnosis and correction by itself.

The goal of our research was to demonstrate the practicality and generality of learning using only analytic techniques combined with outside guidance. Although the demonstrations were performed within the Soar architecture, the results should extend to simi-

lar systems such as Prodigy [Minton *et al.*, 1989] and Theo [Mitchell *et al.*, 1990] that combine problem solving and EBL. These systems may require some architectural extensions, for example, adding the ability to cast operator creation, selection, and implementation as subproblems.

The actual task performed by Robo-Soar was quite simple, and did not address many of the complexities of interacting with external environments, such as dealing with sensor and control errors. Our current goal is to extend Robo-Soar to more complex tasks and expand the spectrum of human interaction. At one end, we plan to investigate refining the guidance so that it is easier to correct and extend the domain theory, approaching the goals of the Instructable Production System where a system is never programmed, only given external guidance [Rychener, 1983]. On the other end of the spectrum, we plan to study experimentation techniques so that Robo-Soar will be able to learn much of the same information on its own, when human guidance is unavailable.

References

[Carbonell & Gil, 1987] J. C. Carbonell & Y. Gil. Learning by experimentation. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 256–266, 1987.

[DeJong & Mooney, 1986] G. DeJong & R. Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176, 1986.

[Doyle, 1986] R. Doyle. Constructing and refining causal explanations from an inconsistent domain theory. In *Proceedings of AAAI-86*. Morgan Kaufmann, 1986.

[Golding *et al.*, 1987] A. Golding, P. S. Rosenbloom, & J. E. Laird. Learning general search control from outside guidance. In *Proceedings of IJCAI-87*, Milano, Italy, August 1987.

[Laird *et al.*, 1987] J. E. Laird, A. Newell, & P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(3), 1987.

[Laird *et al.*, 1989] J.E. Laird, E.S. Yager, C.M. Tuck, & M. Hucka. Learning in tele-autonomous systems using Soar. In *Proceedings of the 1989 NASA Conference on Space Telerobotics*, 1989.

[Laird *et al.*, 1990] J. E. Laird, K. Swedlow, E. Altmann, & C. B. Congdon. *Soar 5 User's Manual.* The University of Michigan. 1990.

[Laird, 1988] J. E. Laird. Recovery from incorrect knowledge in Soar. In *Proceedings of the AAAI-88*, August 1988.

[Minton *et al.*, 1989] S. Minton, J. G. Carbonell, C.A. Knoblock, D.R. Kuokka, O. Etzioni, & Y. Gil. Explanation-based learning: A problem solving perspective. *Artficial Intelligence*, 40(1-3):163–118, 1989.

[Mitchell *et al.*, 1985] T. M. Mitchell, S. Mahadevan, & L. I. Steinberg. LEAP: A learning apprentice for VLSI design. In *Proceedings of IJCAI-85*, pages 616–623, Los Angeles, CA, August 1985.

[Mitchell *et al.*, 1986] T. M. Mitchell, R. M. Keller, & S. T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1, 1986.

[Mitchell *et al.*, 1990] T Mitchell, J. Allen, P. Chalasani, J. Cheng, O. Etzionoi, M. Ringuette, & J. Schlimmer. Theo: A framework for self-improving systems. In K. VanLehn, editor, *Architectures for Intelligence*. Erlbaum, Hillsdale, NJ, 1990. In press.

[Rosenbloom & Laird, 1986] P. S. Rosenbloom & J. E. Laird. Mapping explanation-based generalization onto Soar. In *Proceedings of AAAI-86*, Philadelphia, PA, 1986. American Association for Artificial Intelligence.

[Rychener, 1983] M. D. Rychener. The instructable production system: A retrospective analysis. In *Machine Learning: An Artificial Intelligence Approach.* Tioga, Palo Alto, CA, 1983.

[Segre, 1987] A. M. Segre. *Explanation-Based Learning of Generalized Robot Assembly Plans.* PhD thesis, University of Illinois at Urbana-Champaign, 1987.

Acquisition of Dynamic Control Knowledge for a Robotic Manipulator

Andrew W. Moore
Computer Laboratory, University of Cambridge
Pembroke St, Cambridge, England
Email: awmo@cl.cam.ac.uk

Abstract

To make efficient use of a dynamic system such as a mechanical manipulator, the robotic controller needs various models of its behaviour. I describe a method of learning in which all the experiences in the lifetime of the robot are explicitly remembered. They are stored in a manner which permits fast recall of the closest previous experience to any new situation. This leads to a very high rate of learning of the robot kinematics and dynamics which conventionally need to be derived analytically. The representation is a modified binary multidimensional tree called a *sab*-tree which stores state-action-behaviour triples. This permits fast prediction of the effects of proposed actions and, given a goal behaviour, permits fast generation of a candidate action. I also discuss how the system is made resistant to noisy inputs and adapts to environmental changes. I explain how appropriate actions can be selected in the cases where (i) there has been earlier success and (ii) experimentation is required. This can be used to transform dynamic control to a greatly simplified problem. I conclude with some simulated experiments which exhibit high rates of learning. The final experiment also illustrates how a compound learning task can be structured into a hierarchy of simple learning tasks.

1 Introduction

To make efficient use of a dynamic system such as a mechanical manipulator, the robotic controller needs models of various aspects of its behaviour. Derivation of these models is an issue of central importance in robotics. They can either be hardwired into the controller upon its creation, or else the controller can attempt to learn them itself, using observations from its sensors. The adaptability of the latter method is appealing. Despite this, conventional control has almost invariably used the former method.

It is desirable for a robotic learning system to learn efficiently. Firstly, the information processing needs to take place in real time. Observations about the world need to be recorded and decisions need to be made within the timescale of the robot's dynamics. Furthermore, if learning is computationally cheap then it need never be switched off. Thus the robot can adapt to change throughout its lifetime. Secondly, the rate of learning should be high. This will reduce the expensive (and possibly hazardous) period of time taken up for training. Furthermore, a robot with a high rate of learning can adapt quickly to changes in its environment. The rate of learning depends particularly on the quality of the system's generalization abilities. For a robotic system which learns by monitoring itself there is a second factor crucial to the rate of learning: how and when should experimental actions be applied?

In this paper I discuss an investigation into a practical learning approach for robotic systems which simply uses as its performance element, the explicit set of all the state information it has received through its sensors.

1.1 Manipulator Modelling

The principal example in this paper will be a multi-jointed manipulator. Its controller is able to observe the end-point of the arm by means of a number of cameras connected to a framestore—the robot's retina.

The conventional method first transforms this location according to a *perspective* model into real world coordinates then secondly, via a *kinematic* model, into joint space coordinates. Thirdly, a *dynamic model* of the arm evaluates the effect which gravity and joint torques will have on the joint angles and angular velocities of the arm. The fourth component, the *control* model, uses errors in the perceived location of the arm's end-point to decide which torques need be applied to lessen the error in future.

This approach has been traditionally applied with a fair degree of success [Fu *et al.*, 1987] but has several drawbacks. As well as the expense and complexity, the major problem is adaptability and the dependence upon accurate values of system parameters. Some work has attempted to eliminate the prespecified models entirely by instead making the system learn them. For example, in [Barto *et al.*, 1983] the control and dynamics component were learned, while [Clocksin and

Moore, 1989] investigates the learning of the perspective and kinematic models for a five joint manipulator. In [Mel, 1989] the kinematics and the relationship between the differential kinematics and changes to the explicit retinal image were learned.

2 Proximal Learning

In this work, I investigate a system which learns the perspective, kinematic and dynamics relations in one compound mapping, the *proximal state transition function* (PSTF):

$$(\mathbf{p} \times \dot{\mathbf{p}}) \times \mathbf{t} \rightarrow \ddot{\mathbf{p}} \qquad (1)$$

where $\mathbf{p}$ is the proximal coordinates of the arm on the robot's retina, and $\mathbf{t}$ is the vector of torques applied. As I will explain later, a further result of using this mapping is that the control model can become extremely simple.

2.1 Performance Element

The performance element is a structure which adapts according to a set of *exemplars*—observations of the real world. Each exemplar consists of three vectors of real numbers, called s, a and b. s is an element of the proximal state space—the proximal position and proximal velocity. a is the control action, a vector of joint torques to be supplied. b is the observed behaviour, the proximal acceleration. These exemplars are glimpses of the underlying mapping which we wish to learn:

$$f : \mathbf{S} \times \mathbf{A} \rightarrow \mathbf{B} \qquad (2)$$

The two useful tasks required of the PSTF are:

1. **Prediction:** Given a value $(s, a) \in \mathbf{S} \times \mathbf{A}$, what is $f(s, a)$?
2. **Partial Inversion:** Given a value $s \in \mathbf{S}$ and a target value $b \in \mathbf{B}$, what value of $a \in \mathbf{A}$ (if any) will give $f(s, a) = b$?

This work uses, as its performance element, the explicit set of all observed exemplars. We call this set $\mathbf{E}$. Prediction is based on the *nearest neighbour* by the assumption of smoothness of the relation. To predict $f(s, a)$, find the $(s_i, a_i, b_i) \in \mathbf{E}$ for which (s_i, a_i) is closest to (s, a). The predicted behaviour is b_i. The notion of "closeness" is provided by the Euclidean metric, with the components of s_i, a_i and b_i all scaled uniformly from their maximum ranges to the range $[0, 1]$. In robotics, the ranges of both sensors and actuators are generally explicitly known to the system designer. If not they can be discovered.

Given s and a desired b, partial inversion could be accomplished by finding a $(s_i, a_i, b_i) \in \mathbf{E}$ such that $s = s_i$ and $b = b_i$. However, in general such an exemplar will not be available. Instead, the exemplar with the nearest (s_i, b_i) to (s, b) can be expected to have a good a_i, provided that the predicted value of $f(s, a_i)$ is b_i.

The nearest neighbour generalization has several advantages. The most important is the one-shot learning behaviour. A second advantage is that the generalization adapts to different resolutions of interest, and in particular there is no need to quantize the data. These issues are expanded upon in [Clocksin and Moore, 1989]. The error between the nearest neighbour value and the correct value is expected to be inversely proportional to the local density of exemplars.

The representation of $\mathbf{E}$ should, as well as permitting quick insertion of exemplars, provide fast searches of nearest neighbour in the (s, a)-components and also in the (s, b)-components. This can be achieved using a *sab*-tree. This is a form of a binary multidimensional data structure called a *kd*-tree [Preparata and Shamos, 1985; Omohundro, 1987], modified to permit search on both the (s, a) and (s, b) components as well as efficient smoothing and adaption operations described below. In theory, the number of exemplars examined during a nearest neighbour search tends towards $\log n$ as n, the number of exemplars, gets large. However, the magnitude of n for which this behaviour is reached depends critically on the distribution of the exemplars. Despite this, trials using real data have established that the proportion of exemplars searched is sufficiently low to make nearest neighbour searching in real time entirely practical. Figure 1 shows the average number of nodes inspected against tree size for a 6d-tree of robot dynamics exemplars. The search for the correct point in a *sab*-tree to insert a new exemplar is also $O(\log n)$.

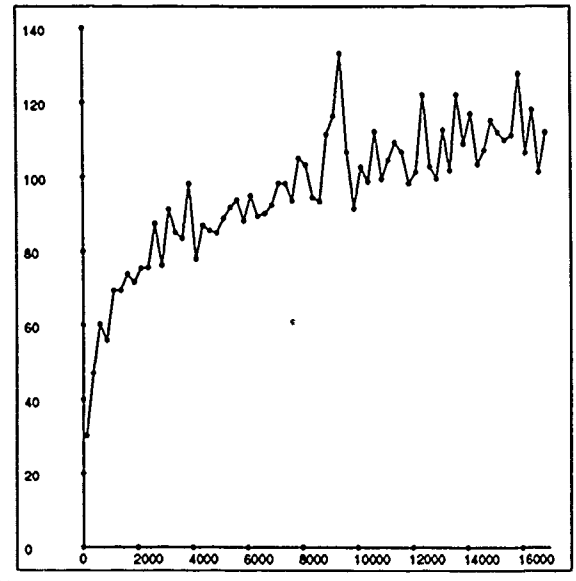


Figure 1 Average number of nodes inspected during nearest neighbour search against tree size.

2.2 Filtering Noise

In practice, the exemplars are of the form $(s_i, a_i, f(s_i, a_i) + e(i))$ where $e(i)$ can be treated as some unspecified noise distribution. The noise can be reduced by taking the weighted mean of several local exemplars. The predicted value at (s, a) is estimated as

$$eval(s,a) = \frac{\sum(weight(s_i,a_i) \times b_i)}{\sum weight(s_i,a_i)} \quad (3)$$

where $weight(s_i,a_i)$ is a function of the distance of (s_i,a_i) from (s,a) which decreases to zero beyond a certain distance r from (s,a). The only exemplars to make a non-zero contribution are those which lie within this distance and so only a local range of the tree needs to be inspected. The value of r is a property of the individual *sab*-tree and can be adjusted according to how much smoothing is required.

To avoid the expense of a range search for each *sab*-tree interrogation, and also to permit partial inversion to make use of the smoothed values, the smoothed values are stored with each exemplar. A node of the *sab*-tree is thus a quadruplet: $(s_i,a_i,b_i,b_i^{\text{smooth}})$ where $b_i^{\text{smooth}} = eval(s_i,a_i)$. Prediction of the smoothed value of an unknown (s,a) is the b^{smooth} component of the nearest $(s_i,a_i,b_i,b_i^{\text{smooth}})$.

2.3 Adapting to Change

The behaviour of a robotic system is really of the following form:

$$f_{\text{actual}} : \mathbf{S} \times \mathbf{A} \times time \rightarrow \mathbf{B} \quad (4)$$

This varies very slowly but unpredictably with time. As time progresses, some of the exemplars will represent inaccurate values. Fortunately, such exemplars can be detected: they are (i) relatively old and (ii) have a large discrepancy between b_i and b_i^{smooth}. These exemplars are deleted from $\mathbf{E}$. The smoothed values of local exemplars are updated to take account of this deletion. The age of an exemplar is recorded by a "date of birth" field. Examples of noise filtering and adapting to change for a one dimensional mapping can be seen in Figure 2 .

It is important that both smoothing and removing inadequate exemplars are computationally cheap, so that learning and adaption will occur *during* the execution of a task. Smoothing occurs each time a point is added to the tree. Obtaining those local points with a non-zero contribution to b^{smooth} costs $O(S + \log n)$ where S is the number of these local points. Each local point has its own b^{smooth} component updated to take account of the new point, and it is then that any elderly, inaccurate points are removed.

3 Using *sab*-trees for Control Choice

In this section I consider how to choose an action given (i) the current state s_q and (ii) the desired behaviour b_q. There are two opposing aims in making a control choice. These are

1. **Perform.** We wish to perform as well as possible given the knowledge contained in the performance element.
2. **Experiment.** In order to perform better in future, it is worth trying actions with no guaranteed success, but with the chance of a valuable discovery.

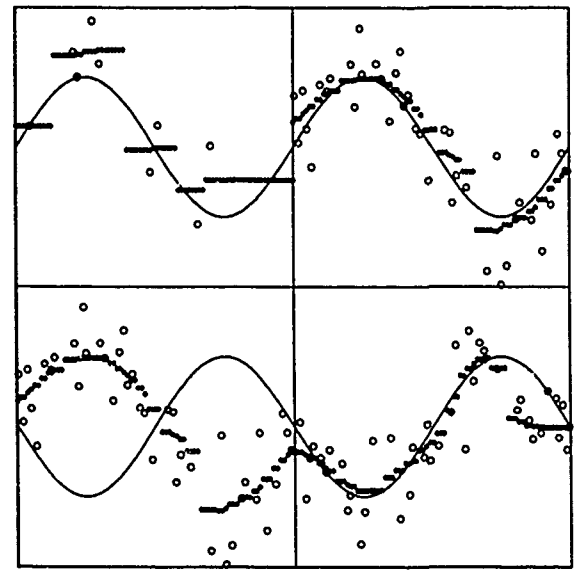


Figure 2: One dimensional SAB-tree, attempting to learn a sine wave from noisy exemplars, shown as white dots. Black dots are the smoothed interpretation. Top left: After 8 random exemplars. Top right: After 50. Bottom Left: The underlying function is changed. Bottom right: After further 50 exemplars.

Aim 1 represents doing the best for the system immediately and aim 2 represents an investment in the future. One approach to this dilemma is to have two modes of operation. The first would be experimental, where actions are chosen randomly with no reference to previous experience. The second is demonstration-mode in which no chances are taken, and the best known action is always used. These modes are altered by an intelligent supervisor who can judge when further experimentation is needed and when it is positively unwelcome. An effective example of these two modes of operation is Mel's MURPHY [Mel, 1989]. This learns the inverse differential kinematics by firstly "flailing" the arm in front of its camera and then making plans based on the results of the observations.

Intermediate schemes are possible in which limited experimentation is permitted by means of limited random perturbations of the best known action, again with experimentation under the control of a supervisor.

For a dynamic system with a large state space the approach of having a controlled experimentation level has some drawbacks. Aside from the requirement for an intelligent supervisor, the major problem is that experimentation is undirected. With a state space of non-trivial dimension, much of the data collected from experiments will be irrelevant to the task, leading to an unnecessarily low learning rate.

In this section I present an action choosing mechanism which is able to take advantage of the explicit exemplar representation to decide, independently of supervision, whether an experimental action is warranted. Given a set of candidate experiments, it also provides an estimate of which is the most promising.

An initial possible action a_1 can be obtained using the partial inversion mechanism described in Section 2.1. The behaviour of a_1 is estimated as that of the closest exemplar to (s_q, a_1). If this predicted behaviour is within τ of b_q, where τ is a task-specific tolerance[1], then a_1 is chosen.

If a_1 is inadequate, then it is necessary to experiment. Instead of making an entirely random guess out of the space of possible actions, the controller makes an educated guess. It generates several alternative random actions, and chooses the one with the highest estimated probability of success. This estimate is obtained by the following heuristic:

$$\text{ProbSuccess}(a) = \text{Prob}(f(s_q, a) \text{ is within } \tau \text{ of } b_q) \quad (5)$$

The behaviour at the unknown point (s_q, a) is modelled as a gaussian random variable with properties depending on the nearest known exemplar $(s_{\text{near}}, a_{\text{near}}, b_{\text{near}}, b_{\text{near}}^{\text{smooth}})$. The expected behaviour is $b_{\text{near}}^{\text{smooth}}$. The standard deviation is proportional to the distance to $(s_{\text{near}}, a_{\text{near}})$. This models the increasing uncertainty as we move further from a known exemplar. The constant of proportionality C reflects how smooth the designer imagines the PSTF function will be.

$$\text{ProbSuccess}(a) = \frac{1}{\sigma}\text{erf}\left(\frac{b_q+\tau-b_{\text{near}}^{\text{smooth}}}{\sigma}\right) - \frac{1}{\sigma}\text{erf}\left(\frac{b_q-\tau-b_{\text{near}}^{\text{smooth}}}{\sigma}\right) \quad (6)$$

Where $\sigma = C \mid (s_{\text{near}}, a_{\text{near}}) - (s_q, a) \mid$. The heuristic is not brittle to the choice of C: the ranking between good and bad candidate actions is changed very little as C varies within a factor of 100.

To illustrate the use of the heuristic, imagine that **S**, **A** and **B** are all 1-d spaces, and that we are in state $s_q = 3.3$ and the desired b_q is 11. Assume further that the only previous experience is that when $s = 3$ and $a = 7$ then $b = 10$. The use of the heuristic, with a tolerance of 0.5, is demonstrated in Figure 3. It compares three possible actions and chooses one which is fairly near to 7, so that the behaviour will be fairly near to 10 (and thus might be close to the goal, 11), but not so near to 7 that the behaviour will be extremely close to 10.

Initially it is likely that for many states the only previous experience will be negative. Then this heuristic favours those actions which are as far away as possible from any yet applied. This is because the probability of success for actions which are close to a previous unsatisfactory behaviour is extremely low, while actions far away have a probability of success which is merely low. Thus a wide variety of actions will be generated until some relatively promising ones are discovered[2].

I have defined a measure for scoring candidate actions, but have not provided a method for obtaining

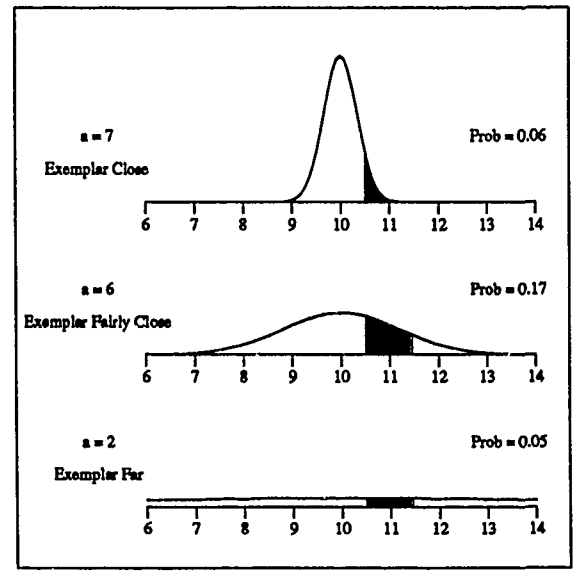


Figure 3: Choosing an action which is likely to cause behaviour in the range 10.5 to 11.5. Top: Action very close to earlier exemplar in which behaviour was 10. Middle: Action fairly close. Bottom: Action very different. The middle case is most promising because the top action is very likely to have behaviour close to 10, and the bottom action could have almost any behaviour.

that which has the highest score. This would require a search of all possible actions, which real time response does not permit. Instead, the favourite of a sample of randomly generated actions is used. A large sample can be expected to provide an action close to that which would be recommended by exhaustive search, but with the penalty of more computation per action. It should be noted that once a successful action has been discovered, computation becomes trivial, as this will be the original action, a_1, returned by partial inversion.

Even if the action is only ever chosen from a small number of candidates, if one performs a series of trials all with the same s_q and b_q, then (subject to b_q being attainable) a_q will eventually converge to a value which achieves the tolerance. However, the state, s_q, of a dynamic manipulator cannot be expected to remain constant. The failure or success of the control choice of the previous time step may no longer be relevant. It is a result of the explicit storage of the exemplars that the performance can nevertheless be expected to improve because information from all those occasions in the learning history which are currently relevant will still be available.

4 Proximal Control

I have explained how the combined perspective, kinematics and dynamics models and their inverses can

[1] τ is not an implicit experimentation level. For example, if high accuracy is required a high tolerance can be specified right from the start.

[2] Notice, though, that if the promising actions were misleading, for example in a local minimum, then upon repeated trials, the heuristic would eventually once again favour further points, because all experiments in the local vicinity would be very close to actions which had failed to meet the tolerance.

be learned, and how the *sab*-tree performance element is used to choose actions given the current state and desired behaviour. Here, I explain how a high level controller for a particular task can make use of these facilities.

Task specifications and also plans to achieve these specifications can take place in the abstract domain of what the robot perceives. It is not necessary to reason in the concrete domain of joint angles, joint torques and feedback gains. Thus, for example, the output of a plan to move the end-point towards an observed goal can simply be that the observed end-point position should start to move towards the observed goal.

Because the underlying model will adapt to changes in the environment, and because the design of the controller is now a fairly simple task (c.f. examples in Section 5), there is less motivation to make the high level controller learn. Instead, at each control cycle, it specifies the acceleration which it would like to be applied to the end-point. Each component of the acceleration vector can be treated separately. Writing $x_i(t)$ as the position of the ith proximal state variable at time t, $v_i(t)$ as its velocity and $a_i(t)$ as its acceleration gives the trivial control equations:

$$v_i(t_1) = v_i(t_0) + \int_{t_0}^{t_1} a_i(t)dt \quad (7)$$

$$x_i(t_1) = x_i(t_0) + \int_{t_0}^{t_1} v_i(t)dt \quad (8)$$

Examples of this sort of controller are given in the following section.

5 Experimental Results

These experiments use a dynamic simulation of a planar two jointed robot arm moving under gravity. The dynamic model is from [Fu *et al.*, 1987].

5.1 Tracking a moving point

The task is to track a point moving at constant speed along an anticlockwise circle in proximal coordinates. Figure 4 shows the initial state of the arm, and the target trajectory.

At each time step we can observe the position and velocity of the image of the arm's endpoint. From the task specification we can obtain the ideal image position in two time steps. We consider each variable separately. Let x_0 be the perceived current position of one such variable, and v_0 its perceived current velocity. If we apply constant acceleration a_0 for one time step, followed by acceleration a_1 for the next, then from Equation 7 we obtain the perceived position and velocity in two time steps:

$$x_2 = x_0 + 2v_0 + \frac{1}{2}(3a_0 + a_1)$$
$$v_2 = v_0 + a_0 + a_1 \quad (9)$$

We insert the ideal x_2 and v_2 to obtain the ideal acceleration:

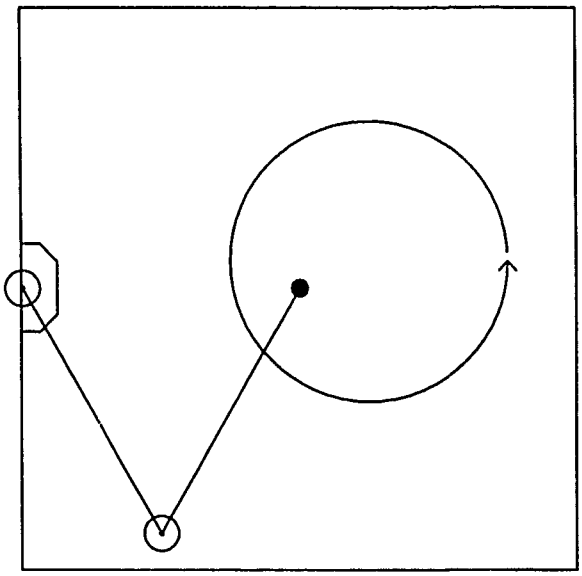


Figure 4: Two jointed planar arm acting under gravity. Its task is to follow the circular trajectory.

$$a_0 = x_2 - x_0 - \frac{1}{2}(3v_0 + v_2) \quad (10)$$

The ideal acceleration is computed for each variable independently. The proximal acceleration vector is the behaviour b_q which is passed, along with the current proximal state s_q, to the *sab* control choice mechanism of Section 3. The resulting a_q is a torque vector which, it is believed, will produce the proximal acceleration. If these torques are too large, the largest torque in the same direction is applied. After these torques have been applied the actual proximal acceleration is observed. A new exemplar consisting of the previous state, the torques and the actual acceleration is added to the *sab*-tree.

Figure 5 is of the first nine journeys around the circle. At the start of the first journey there is no knowledge of the PSTF. It does not take very long at all before some very rough control is established, tending at least to provide thrusts in the correct proximal direction, if not accurately. But within only a very short time it in fact remains very close to the target point.

Figure 6 is of the first nine journeys around the circle with a substantial noise component added to the arm dynamics. The rate of learning is slower because the initial exemplars are misleading but eventually the exemplars become sufficiently numerous that the smoothed values represent the ideal behaviour. The performance will never reach that of the original example, because random noise is always being added.

Figure 7 demonstrates how the controller adapts when the arm dynamics change suddenly during the fourth circuit[3]. Initially the performance is disastrous, but it gradually recovers. It takes several cycles to re-

[3]The actuator at joint two suddenly starts providing twice the original torque

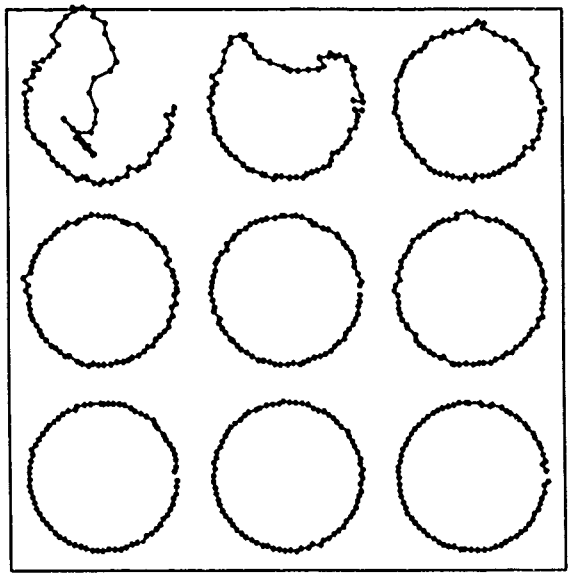


Figure 5: Trajectory of arm during first nine circuits, read from left to right down the figure. The hand was initially near the centre of the circle.

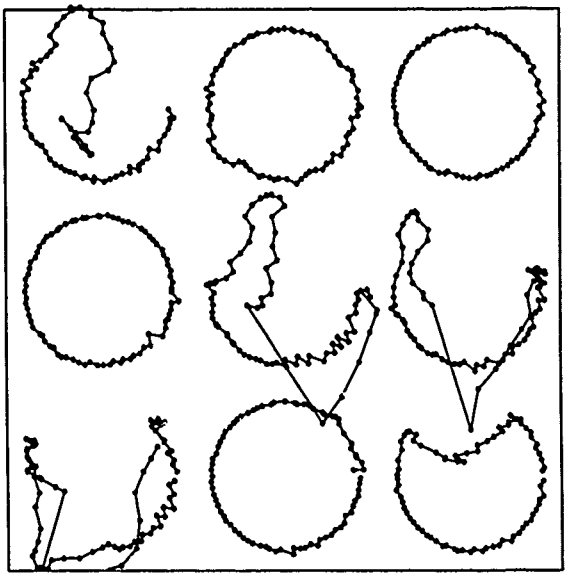


Figure 7: Trajectory of arm. Manipulator dynamics change during the fourth circuit. By the eleventh circuit (not shown) the control was entirely recovered.

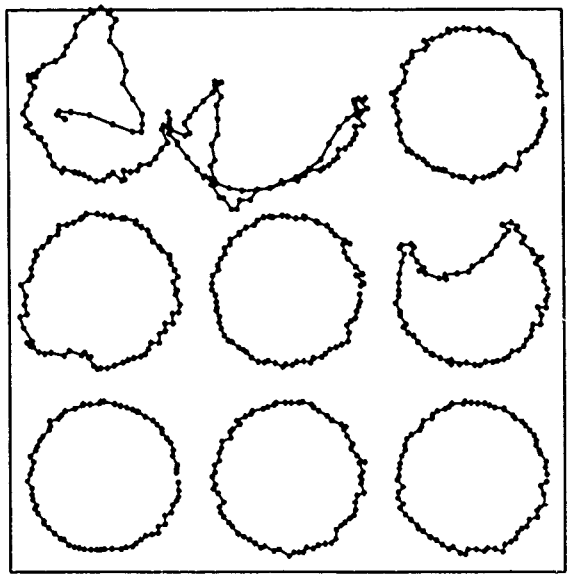


Figure 6: Trajectory of arm with substantial random noise added to the manipulator dynamics.

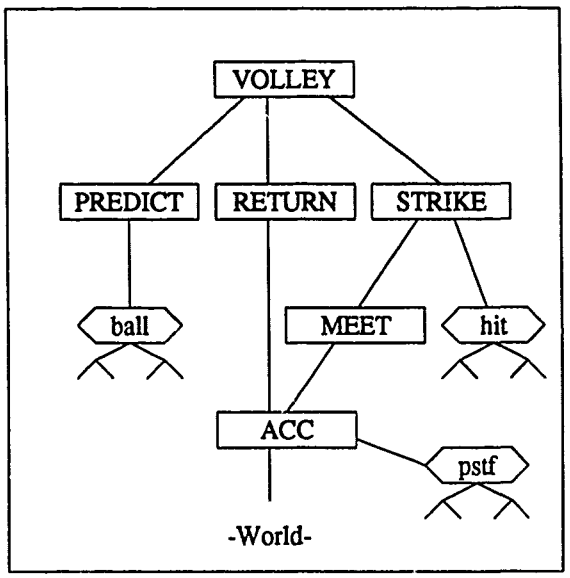


Figure 8: Structuring the Volley task. The tasks are shown in rectangles, *sab*-trees are shown in ovals.

cover because the disasters mean that it doesn't immediately re-experience all the areas near the trajectory.

Each experiment took less than 10 minutes real time for the simulation and learning.

5.2 Underarm Volleying: A Compound Task

For this task, the simulated arm is given a bat which is fixed at right angles to the arm's second linkage. A ball is fired towards the arm. The task of the arm is to volley the ball so that it lands in a bucket placed nearby. The visual location of the bucket and ball can be obtained.

This complex task can be structured into the hierarchy of subtasks shown in Figure 8.

The overall task, called `Volley`, is broken into three subtasks. `Predict` estimates the ball's behaviour. `Strike` brings the bat to contact the ball at a controlled position and speed. `Return` holds the bat steady after the strike.

The `Predict` task models the perceived behaviour of the ball prior to being hit. This model is learned by a *sab*-tree, which estimates the time until the ball arrives within range of the arm and the state of the ball at this point. It is a mapping

$$\underbrace{\text{Ballstart}}_{S} \times \underbrace{()}_{A} \rightarrow \underbrace{\text{Ballhit} \times \text{Timehit}}_{B}$$

which is updated once every trial. There is no control over this aspect of the ball's behaviour so **A**, the space of control actions, is empty.

The **Return** task simply computes a proximal acceleration to thrust the endpoint towards the stationary waiting position. The torques to achieve this acceleration are computed and executed by the low level **Acc** task.

The **Strike** task controls the ball indirectly by means of a collision between the ball and the bat. It requires a model of the real world:

$$\underbrace{\underbrace{\text{Ball at Hit} \times \text{Bat position and direction}}_{S} \times \underbrace{\text{Bat speed} \longrightarrow}_{A}}_{\text{X coordinate of Landed Ball}} \quad (11)$$

This too can be learned using a *sab*-tree. From trial to trial, the **Strike** task attempts to always position the bat to contact the oncoming ball at the bat's centre, and to have it moving in the same relative direction at impact. The speed of the bat at impact is varied. This speed affects the landing position of the ball, and so can be used to indirectly control the landing position. At the trial start the *sab*-search is given s_q as the estimated ball state when it arrives in range and the intended bat position and direction. The search produces a recommended speed for the bat at impact.

It should be noted that this *sab*-tree, like the others, is simply a set of objective observations about the world, and its own accuracy does not depend on the performance of the subtasks which are being learned. There is no blame or credit assignment problem. For example, suppose that we believed that hitting speed S_1 at position P_1 would ensure that the ball landed at X_1, but due to a low level error the ball was volleyed with the correct speed S_1, but at the wrong position P_2. The ball then lands at X_2. The speed S_1 will not now be wrongly associated with landing at X_2: the *sab*-tree will simply contain an observation that hitting with speed S_1 at the wrong position, P_2, results in a landing at X_2, and will contain no explicit prediction as to what would happen were the ball hit at the correct position.

The **Meet** task guides the initial proximal state to the target impact state. Working in the proximal space, it can do this for each state variable independently. The ith variable has a current state $(x_i(0), v_i(0))$ and the controller must invent a sequence of accelerations so that at the predicted time t_{goal} of the ball's arrival the state will be $(x_{\text{goal}}, v_{\text{goal}})$. In fact this is easy. Consider accelerating with acceleration a_0 for t_1 seconds, then with acceleration $-a_0$ for t_2 seconds. Then there are three equations in three unknowns: Equations 7 and 8 with constant acceleration and the constraint that $t_1 + t_2 = t_{\text{goal}}$. The unknowns are a_0, t_1 and t_2. Finding the value of a_0 to achieve this is straightforward scalar algebra, which amounts to solving a quadratic equation. The ideal proximal acceleration can thus be recalculated on every time step to account for earlier errors.

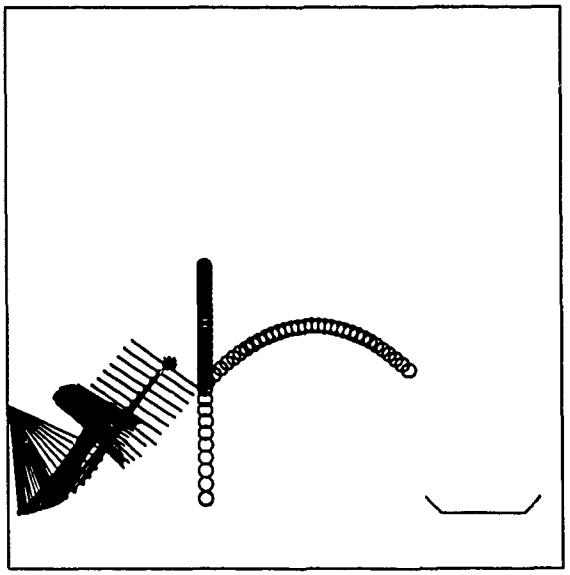


Figure 9: An early volley attempt. The ball is fired from the right towards the arm. From the initial state of the ball the controller erroneously predicts it will arrive at the grey circle. The ball is clipped by the bat and flies up vertically before falling to earth.

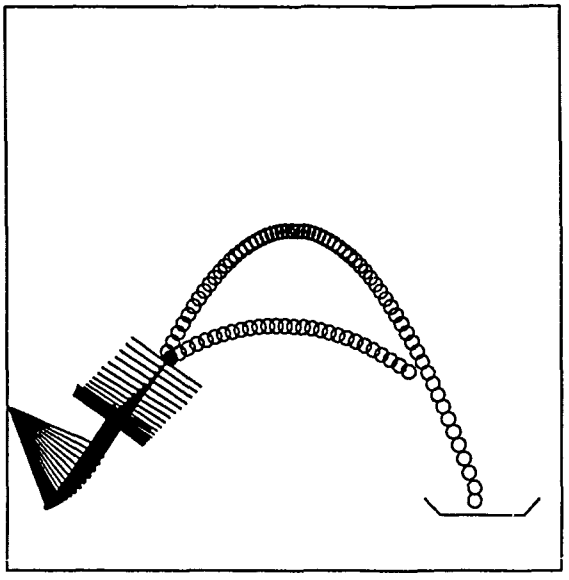


Figure 10: A successful volley. As well as modelling its own arm, the controller has correctly predicted where the ball will be when it is in range and found a correct speed with which to hit the ball back to the bucket.

The **Acc** subtask is the task used in the circle tracing experiments to derive a torque from the PSTF which achieves the requested proximal acceleration.

Figures 9 and 10 show the behaviour of the arm during an early and late trial respectively.

Figure 11 graphs the performance of a relatively easy task, where the ball is always fired from the same position and velocity and the bucket always remains in the same place. After five trials a suitable hitting speed is discovered, and a value close to this speed is used for subsequent trials.

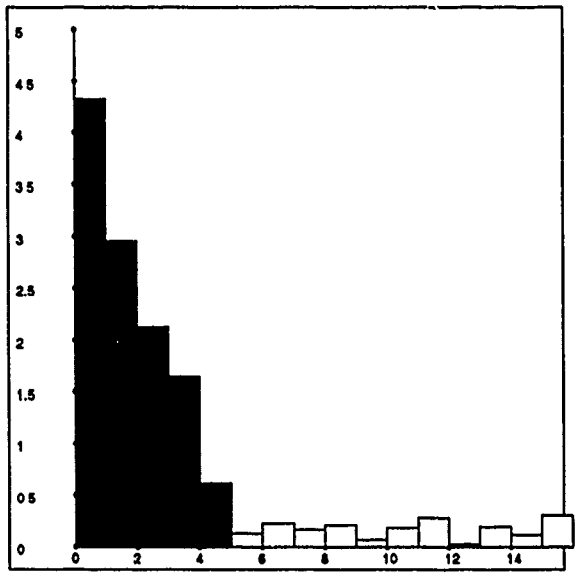


Figure 11: Histogram of distance from bucket against trial number. The successful volleys, which landed in the bucket, are shown in white. During these trials the bucket was fixed and the ball always fired with the same speed and direction.

Figure 12 displays the results of the considerably harder task in which the ball is fired at a random speed for each trial, and the bucket is placed in a random position. It requires approximately twenty trials before the behaviour can be said to be fairly skilled.

Figure 13 shows the behaviour when the bucket is placed randomly and the ball is fired with a random speed *and* direction. There is an improvement in behaviour but the probability of failure is still roughly 20% even after over 100 trials. This is because even then there is still a fair probability that the starting state of the ball is sufficiently far from any previous experience that the behaviour predicted by the nearest neighbour is inadequate.

6 Conclusion

The experiments in the previous section all took only a small amount of real time. Both the rate of learning, and the information processing were fast. For example, in the circle tracking task, after only twenty state–action–behaviour observations, the arm was already under some sort of rough control[4]. The primary reason for this is that only one presentation of a sample of data is required for the knowledge contained in the data to be fully represented. This can be contrasted

[4]The direction in which the hand was accelerated was already on average within 30 degrees of the direction requested by the proximal controller.

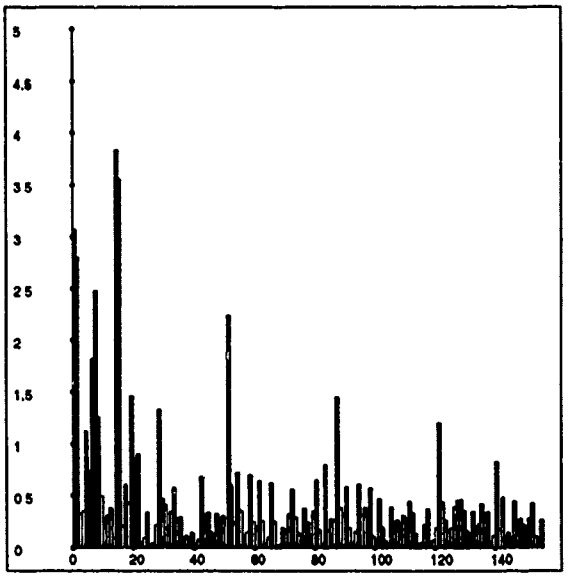


Figure 12: Histogram of distance from bucket against trial number. Before each trial the bucket was placed at a random position. The ball was always fired with random speed but constant direction.

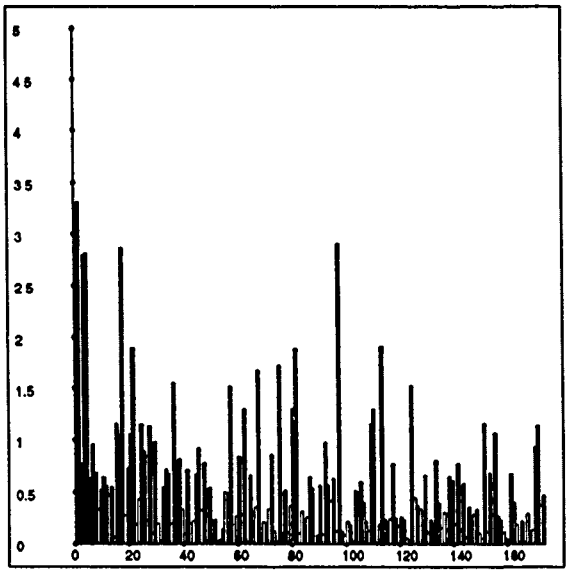


Figure 13: Histogram of distance from bucket against trial number. Before each trial the bucket was placed at a random position. The ball was always fired with random speed and random direction.

to an approach in which the knowledge is represented indirectly, perhaps as a set of weights in a network. When each exemplar is presented, all or some of the weights are modified according to a rule which will only eventually, on repeated future presentations of similar exemplars, converge to a representation of the same knowledge.

A second reason for the fast rate of learning is that the mappings which are learned are objective observations about the real world, rather than task-specific knowledge of which responses are "right" or

"wrong". This means that the knowledge obtained from a "wrong" response in one context, might in another context or task be useful positive information. For example, when an erroneous volley misses the target, the controller has the consolation that should the target, in a similar situation, ever be near the point at which the ball in fact landed, it will know about a potentially successful volley.

The performance element, the *sab*-tree, was updated and interrogated in real time. The small overall times for the experiments indicate that the speed of nearest neighbour searching is adequate. The exemplars obtained when performing a task tended to be distributed extremely unevenly, which helps the nearest neighbour search considerably. With the decreasing cost of memory and relatively fast processors it should be entirely practical to search trees in the order of a million exemplars. This is the size of a tree obtained from monitoring a robot constantly for over eleven days with one observation a second. Furthermore, it is likely that the size of the trees could be reduced an order of magnitude by only storing those exemplars which were predicted incorrectly prior to their observation, but this has not been investigated.

As a result of being able to reason in proximal space, the low level control can become straightforward. The behaviour of the controller becomes easier to understand when tasks are specified in relatively abstract terms such as "move up" or "accelerate towards this point", instead of concrete joint space terms. Just as a compiler allows a programmer to think in abstract terms without needing to know about the underlying machine language, so *sab*-tree learning permits the system designer to devise control strategies without needing to know about the specific parameters, geometry or dynamic behaviour of the robot, even if these may change over time.

By structuring a more complex task as a hierarchy of subtasks, the higher levels of the controller can also become relatively easy to implement. When models of the real world are required for these higher tasks, *sab*-trees can once again be used. However, the structuring needs to be performed by some expert, presumably human. In order to be able to classify the controller as truly autonomous, it would have to be able to devise the structuring strategy itself. At this stage the lack of full autonomy is a compromise, justified by the observation that a large proportion of the effort in robotic control using conventional methods is not in inventing abstract strategies, but in modelling the world.

Acknowledgements

I would like to thank my supervisor William Clocksin for the initial motivation for this work, and for his help and advice. Thanks also to Thomas Clarke, Barney Pell and Thomas Vogel. This work is supported by a grant from SERC.

References

[Barto *et al.*, 1983] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike Adaptive elements that that can learn difficult Control Problems. *IEEE Transactions on Systems Man and Cybernetics*, 1983.

[Clocksin and Moore, 1989] W. F. Clocksin and A. W. Moore. Some Experiments in Adaptive State Space Robotics. In *Proceedings of the 7th AISB Conference, Brighton.* Morgan Kaufman, April 1989.

[Fu *et al.*, 1987] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee. *Robotics: Control, Sensing, Vision and Intelligence.* McGraw-Hill, 1987.

[Mel, 1989] B. W. Mel. MURPHY: A Connectionist Approach to Vision-Based Robot Motion Planning. Technical report ccsr-89-17a, University of Illinois at Urbana-Champaign, June 1989.

[Omohundro, 1987] S. M. Omohundro. Efficient Algorithms with Neural Network Behaviour. Technical report uiudcs-r-87-1331, University of Illinois at Urbana-Champaign, April 1987.

[Preparata and Shamos, 1985] F. P. Preparata and M. Shamos. *Computational Geometry.* Springer-Verlag, 1985.

Feature Extraction and Clustering of Tactile Impressions with Connectionist Models

Marcus Thint and Paul P. Wang
Department of Electrical Engineering
Duke University

Abstract

We study the feasibility of adaptive pattern recognition of robotic tactile impressions using connectionist models. This paper presents interim simulation results of coupled back-error propagation (BEP) networks that (i) extract relative gradient features via data compression, (ii) clusters families of grey-scale patterns constrained by geometry, size, and activation levels, and (iii) classifies these surface profiles to pre-specified categories. The constraints imposed on the training data are designed to capture the essence of tactile patterns and force the artificial neural systems (ANS) to extract useful features. Receptive field (rather than fully connected) processing units are used to encode subtle features among their activation patterns. This work initiates ANS applications in the tactile domain and reveals basic characteristics of BEP networks to highly constrained training data.

1 Introduction

Numerous techniques are well known for pattern classification *given* an appropriate set of input features, but the *determination* of such features remain the central challenge in many pattern recognition processes. Efficient solutions for automatic (machine) derivation of these features have been elusive. We explore the possible benefits of using artificial neural systems (ANS) for feature extraction, clustering, and classification of relative gradients within 2D tactile impressions.

In robotic applications, tactile sensing is critical when a machine is required to perform dexterous manipulations and precise/fragile assembly tasks. Also, for object detection or identification purposes tactile sensors can complement or substitute for more costly vision sensors [whenever contact can be made]. Presently, a variety of tactile sensor technologies exist: conductive elastomers, ferroelectric polymers, optoelectronic sensors, and silicon strain gauges [Nicholls and Lee, 1989]. Reliable sensor data can be obtained from some of the industrial-quality products. Sensors vary in size, shape, and resolution for mounting on the "fingers" of a robot or atop a work-surface.

Research and development in tactile sensing comprises: (i) biological/physiological studies, (ii) design of artificial sensors, (iii) planning and control of haptic perception to acquire object data, and (iv) pattern recognition of the tactile data. The work described below is restricted to topic (iv) – processing and classification of tactile impressions. Past works in this area have primarily been limited to applying "borrowed" algorithms from image processing to the tactile domain. These methods work well for classification of simple, binary silhouettes, but are inadequate for real-time applications in more complex domains. We begin by investigating the domain of a planar sensor matrix and techniques for processing its data in the connectionist realm.

2 The Training Data

True tactile patterns comprise both the geometric shapes of the object as well as the surface contours derived from 3D force and torque distributions on the sensor. Although specific sensor designs may vary, each 'tacel' provides at least the normal($\hat{z}$) force reading, and some advanced designs yield all 3D force components. The force experienced by each tacel is converted into a grey-scale number, and the collection of values from all tacels form the "image" to be analyzed. Embedded information from such data, possibly combined with the global 3D moments experienced by the sensor, must be manipulated to derive a "sense of touch" in robotic applications.

We begin with a small training set based on nine surface contours depicted in Figure 1. When pressed onto a compliant tactile sensor, these rigid surface

profiles would produce the corresponding impressions shown in Figure 2. There are six surface profiles of rectangular silhouette and three of circular silhouette named and tagged as: *bar(B), rod(R), wedge(W), slant(L), bump(U), hole(H);* and *sphere(S), cylinder(C), cir_slant(I)* respectively.

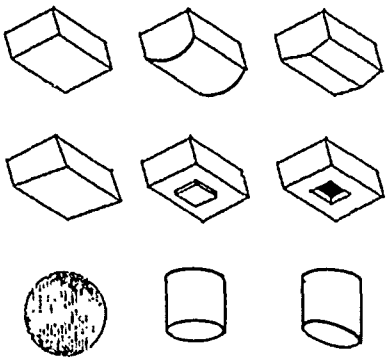


Figure 1: Surface Primitives

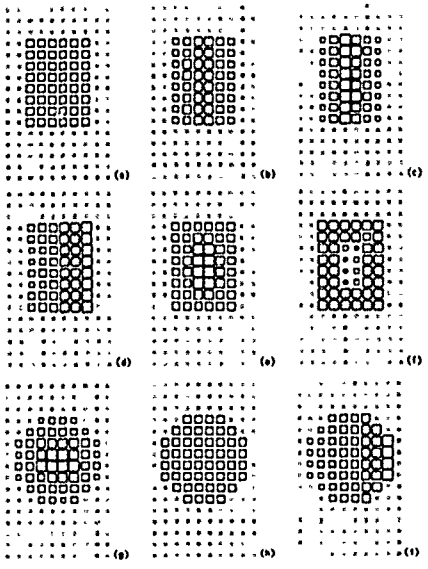


Figure 2: Corresponding Tactile Impressions

The sensor response reflects Lord Corporation's LTS200 model tactile sensor [Rebman and Trull, 1983; Muthurkishnan *et al.*, 1987], but similar data can be obtained from other high quality, robust sensors. The impressions in Figure 2 assume uniform pressure distribution on the solid objects, and that the entire bottom surface of the object is in contact with the sensor. The squares are graphical icons representing tacels, and their sizes are directly proportional to the amount of normal force experienced at each site.

Within each data group (rectangular or circular) the size and shape of the contact surfaces are identical so that the ANS must discriminate the patterns based on intensities of the force experienced by the tacels rather than the object's size or geometry. Moreover, the 'global' or ensemble force experienced by the matrix of tacels are held approximately constant in order to force the ANS to distinguish the relative gradient patterns rather than the overall pressure on the sensor. These constraints are important to ensure extraction of useful features, but do not imply that *all* tactile data embody such characteristics; we study the more restrictive and difficult case and contend that relaxing any one of the constraints (eg. different geometries) would pose a simpler feature extraction problem. Previous works in tactile pattern recognition described processing of binary patterns and/or differing geometries which do not focus and capture the essence of tactile data. [Hering *et al.*, 1990] [Muthurkishnan *et al.*, 1987] [Marik, 1981] [Kadonoff, 1983] [Togai, 1982] [Hillis, 1982] [Sato *et al.*, 1977] [Takeda, 1974] [Kinoshita, 1973]

3 Network Description

In this study, the domain of ANS is a controlled environment such as a robot workcell in which several regular objects are assembled with the aid of a planar tactile sensor. Thus, the ANS are not required to create categories from random, arbitrary input patterns; in fact, during training, the desired output categories would be specified and associated with a subsequent task. As a result, a more direct architecture of a layered feed-forward back-error-propagation (BEP) network is chosen rather than one based on Adaptive Resonance Theory. The 'standard' BEP network is a nonparametric classifier, and its algorithm is based on minimizing an error function $E_T(W) = \sum_p E_p(W)$ via recursive computation of the error signal δ with gradient descent in weight-space [Rumelhart *et al.*, 1986]. The simulations were performed using the Rochester Connectionist Simulator [Goddard *et al.*, 1989].

We first study the problem of processing static tactile impressions from one planar sensor matrix, and at this juncture, assume that object impressions are totally contained within the sensor. It is desirable to achieve pattern recognition invariant to translations and rotations of the tactile impressions. (An object of different scale could effectively be of another class, since its handling and subsequent operations may differ.) In this initial phase, we impose only the translation invariance requirement and use a 2D Fourier transform as a preprocessor. The network architecture is shown in Figure 3.

The first BEP module extracts salient features among the training patterns and the second module forms decision boundaries between them for catego-

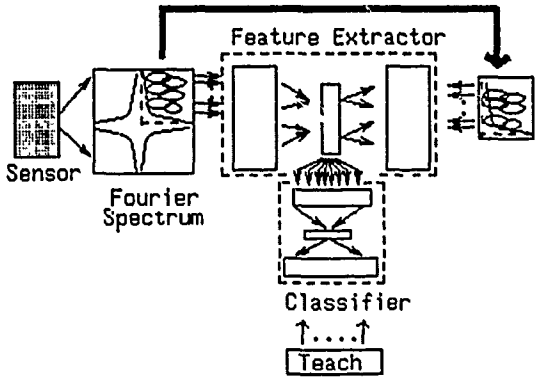


Figure 3: Network Architecture

rization. The ovals in the Fourier spectrum are to indicate that receptive fields units are used rather than the standard fully connected BEP scheme. The input to the feature extraction network is quadrant-one of the corresponding object's Fourier spectrum as shown below.

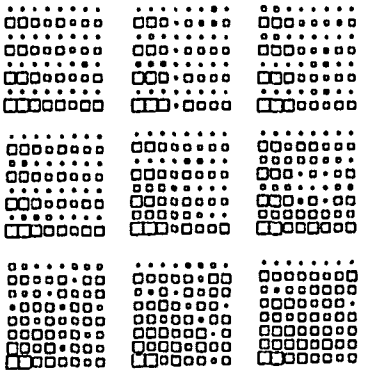


Figure 4: Input Vectors: Partial Fourier Spectra

The spectra of the rectangular patterns (top two rows in Figure 4) and the circular patterns are distinctive as a group indicating that differences in geometry pose a relatively easy classification problem. Within each group, however, the distinguishing features are more subtle and pose a challenging learning problem for the ANS. Since the spectra of high frequency terms decrease rapidly, the function $D(u,v) = \log(1+|F(u,v)|)$ is used instead of $|F(u,v)|$ to display the image and preserve the zero values in the frequency plane. The FFT output yields translation invariance, since its magnitude $(F(u,v)e^{-2j\pi(ux_o+vy_o)/N})$ is not affected by a shift in the time domain.

The hidden layer in the feature extraction module receive input from a small neighborhood of input units. The exact number of hidden units would depend on the size of the input vector and the receptive fields; we find that the more similar the input vectors are, smaller receptive fields are required to encode minor feature differences, and thus more hidden units would be created. The connectivity issue is discussed further in Section 4.2. The output units receive the same information and connectivity pattern as the input, since the feature extraction BEP network receives no external teach vector.

4 Feature Extraction

Saund [Saund, 1986] presented some theoretical background for using connectionist networks to discover constraints in multidimensional data via dimensional reduction. A subset m-dimensional features embedded in n-dimensional data source can be abstracted into hidden units of a BEP network. Cottrell et. al [Cottrell *et al.*, 1987], and Kuczewski et. al [Kuczewski *et al.*, 1987] reported on using BEP network as a self-organization structure in image compression/data reduction. The feature extraction module in Figure 3 is based on these same principles and background concepts are omitted here, except to reiterate that no explicit external "teacher" is used in this scheme; the network maps input patterns onto themselves while performing data compression. In the initial simulations, the feature extractor module compacted salient features among ten 64-dimensional input vectors into 8 feature (hidden) units. Kuczewski demonstrated more significant reduction ratio from 255 to 3 dimensions in a four-layer network; here, the specific constraints of the input data in the tactile domain are of interest to us. Discrimination of tactile impressions pose an interesting task for the ANS, and empirical results indicate that (i) ANS abstract the simplest (but not necessarily useful) criteria while encoding features, unless proper constraints are imposed on the input data, and (ii) the standard fully-connected paradigms of the BEP architecture may not necessarily yield useful data representation.

4.1 Constraints Imposition

As mentioned in Section 2, the patterns in the training set are constrained to have the same (i) geometry (ii) size, and (iii) total force applied to the sensor. The intent is to make the feature extraction module determine that *relative force gradients* are the key features to be learned, by repeated presentation of the training set. However, the features it does (or does not) capture can be deduced by testing the network in its "trained" state on new data. For example, if the training set itself were similar to that in Figure 2, but the "footprint" of the *rod* differed in shape to the *bar*, and the *cylinder* was of different size than the *sphere* (all with proper pressure distributions) the network would err and exhibit marked sensitivity to

changes in geometry and size of these objects [1] A less conspicuous feature is the aggregate force applied to the sensor. If the shape and size are constrained but the total force applied (reflected by the sum of grey-scale values across the sensor) is different, the network will use the ensemble force as the discriminating factor among those patterns. Hence, the network misclassifies when the same gradient profile with different total force is shown to it. Note that although the extracted features are found by the network and *not* "human-engineered", the construction of the training set from which the module learns must be carefully crafted. We conjecture, however, that if large numbers of examples with varying geometry, size and forces are shown to the network for *each* of the surface profiles, and the network is allowed to evolve over a 'long' period of time, it will discover that relative force gradients are the common features. However, for initial focused studies, system parameters must be carefully controlled.

4.2 Connection Patterns

Even if proper constraints are imposed on the training set and the network is trained to extract the salient force gradients, the feature information may not be accessible for training subsequent modules. That is, we want the abstracted features to be manifested as activation patterns across the internal hidden layer such that these patterns can be used as the training vectors for the classification module. During self-organization, we compute the average differences (absolute or squared) over all pixels in a given image and compute one metric per training pattern. This value is then compared with a permissible error ($PErr$) term that determines when to terminate training. Using the conventional fully-connected, feed-forward BEP network architecture, repeated runs consistently resulted in a failure to encode salient features as activation levels on the hidden units. The networks do converge below $PErr$, but all the feature information is stored among the pattern of link weights rather then as activation levels of hidden units. A typical convergence pattern is shown below in Table 1.

Table 1: Typical Fully-connected Convergence

Hidden Activations (all 9 patterns)							
.07	.06	.00	.07	.02	.06	.07	.05

The activation pattern across all 8 units are the same regardless of the input pattern, thus it is impossible to discern input patterns from this data. The mean-absolute and mean-squared differences for all patterns, however, indicate that the reproduction at the output layer was quite faithful to the input pattern. Nevertheless, this result is unacceptable since training vectors for the next module are unavailable. We thus employ the concept of receptive field neurons. With partial-connections from input to hidden units with a neighborhood radius of k units and center-to-center distance of l units, the typical convergent hidden layer activations are as shown below in Table 2.

Table 2: Convergence with Receptive Field Units

	Hidden Activations							
B	.00	.03	.02	.93	.01	.04	.50	.11
R	.01	.02	.02	.96	.01	.08	**.38**	.04
W	.06	**.23**	.02	.96	.01	.07	.54	.07
L	.05	.07	.049	**.65**	.03	**.23**	**.62**	**.30**
U	.04	.09	.05	.63	**.11**	.25	.63	**.14**
H	**.11**	.05	**.18**	.50	.07	**.62**	.60	**.45**
S	.01	.23	.13	.50	.12	**.42**	.65	.32
C	.02	.05	.021	**.41**	.09	**.52**	**.74**	.07
I	.19	**.17**	.05	.66	.09	.24	.63	.14

The features among the patterns are not distinctive such that one is "high" while others are "low", but results indicate that based on even a small threshold θ, some combination of the 8 feature units can distinguish one surface profile from any other one. In the constrained tactile domain, the networks do not have a wide dynamic range to operate near boolean limits; the differences between the data are subtle, and accordingly, feature units converge at various analog values in its activation range. Preliminary results indicate that the dimensionality of the feature space n is approximately $0.7 < n \leq p$ where p is the number of pattern classes. During fully-connected configuration, each hidden unit is receiving input from the entire input pattern of N units and is imposed with extraneous/redundant data. With partial connectivity, not only is the amount of data reduced to some proportion of $\frac{k^2}{N}$, but more significantly, each unit receives a fraction of k^2 private segments of the each image. The ratio of $\frac{l}{k}$ ($\approx k < l < 2k$) controls the amount of overlap between the receptive fields, but simulations show that it does not have appreciable affect on feature extraction properties. The important factor is the segmentation of the input data by receptive fields to enable the network to discover small distinctions among the training patterns.

A design question arises as to how many feature (hidden) units are required and how large the receptive fields should be. In general, the number of required

[1] Note that for robust performance, a comprehensive training set *should* include corresponding sets of each object in various shapes and sizes, but this initial training "subset" suffices for preliminary studies.

features nodes are not known a-priori, and it will depend intimately on the degree of correlation of the input vectors. To design a feature extractor network without too much human preprocessing and scrutiny, a network can start with a large number of feature nodes, each with small receptive fields. After the network converges, the activation patterns across the feature units separate crucial from extraneous units. The extraneous units can then be "pruned" either by human-inspection or some automatic algorithm as discussed by Sietsma and Dow [Sietsma and Dow, 1988]. The module described above started with 16 hidden nodes; it turns out that only 8 were sufficient to distinguish between the patterns.

5 Clustering Groups of Patterns

Thus far, we discussed two coupled BEP networks, that can extract features and classify the basic data set of Figure 2. Next, we extend the problem and test if these ANS can categorize "families" of such patterns. We relax the previous stipulations of uniform pressure distribution on the objects and consider some distorted patterns. In advanced robotic applications where the robot must operate without precision peripheral devices such as positioning jigs, consistent acquisition of flawless sensor data is highly improbable. On a tactile sensor, for instance, an object is likely to be pressed slightly harder on one section than another.

Hence, the training set is augmented to include additional impressions of the nine objects as they are "rolled" slightly to the left, right, top, and bottom of the objects. We simulate up to 20 % pressure gradients for four basic directions (right,left,top,bottom) on the sensor, and obtain a total of 45 training patterns. The impressions for the *bar* in various "rolls" are shown in Figure 5.

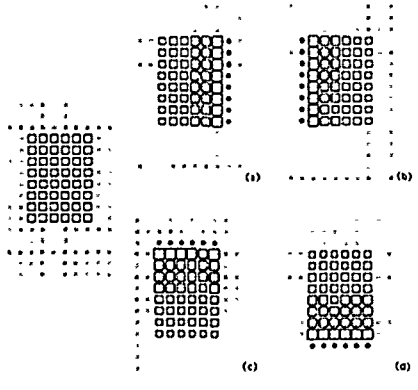


Figure 5: Distorted 'bars' to Cluster

Despite the distortions, the network should learn that all five patterns belong to class *bar*, and similarly, the other pattern clusters to their respective categories. The problem becomes much more difficult, because the distinctions between the surfaces become less clear, and one pattern can resemble another very closely. For instance, the impression of the *bar* rolled to the right (Figure 5a) is now *very* similar to the original *slant* (Figure 2d) pattern. Only subtle difference remain between the patterns, such as the slightly activated neighboring column of tacels to the right edge in Figure 5a, resulting from increased deformation of the sensor surface there. Likewise, when the *slant* is rolled to the left, it resembles the original bar (Figure 2a), and the same for the *cylinder* and *cir_slant* objects. Note that the corresponding Fourier spectra that serve as the input vector also becomes more and more similar, and poses a difficult learning problem for the ANS.

This problem requires the ANS to cluster the five patterns as a group and derive cluster centers that are as distant as possible to distinguish between the clusters. Clustering process can be view as *unsupervised classification* in which N samples (each characterized by an n-dimensional vector) are classified into M classes $(\omega_1, ...\omega_M)$. The classification Ω is a vector made up of the ω_ks and the configuration X^* is a vector make up of X_is. The cluster criterion J then is a function of both Ω and X^*;

$$J = ([\omega_{k1}, ...w_{kN}]^T; [X_1^T, ...X_N^T]^T)$$

In seeking data cluster via standard algorithms, some measure of similarity such as J establishes a rule for assigning patterns to the domain of a particular cluster center. Moreover, since the proximity of two patterns is a relative measure of similarity, it is usually necessary to establish a threshold in order to define degrees of acceptable similarity in the cluster-seeking process. Preliminary indications are that the feed-forward BEP network could prove to be quite effective in forming both the similarity measure and the threshold via some combination of its link weights and bias parameters.

Simulation results indicate that for the clustering problem, more hidden units are significant and contribute to the feature extraction process. The activation levels of the receptive field hidden units contain the feature information. However, the correlation of the input patterns are high and there exist many overlaps of features among the 45 patterns. Even after the feature extractor converges with low *PErr*, it is difficult to discern some of the boundaries due to high dimensionality of the feature space. As one may suspect, clear boundaries can be found among the extrema of many hidden nodes that separate the circular and rectangular silhouettes, but as little as one hidden node separates some patterns of same geometry, such as (*bar,rod*), (*rod,wedge*), and (*cylinder,cir_slant*). The collection of all the features, distinctive and subtle, serve as training vectors for the classification module.

6 Pattern Classification

The classification module is a very standard, fully-connected two-layer BEP module and its details are omitted here. The values for its input units are propagated from the activation levels of the feature nodes from the preceeding module. The output units classify them into nine categories, this time supervised by an explicit teach vector. For the basic pattern set, using the patterns shown Table 2 above, the classifier learns to separate those features, and achieves clean and distinct class separations at convergence. Similar results are obtained for classifying all 45 patterns into nine categories, although convergence time (number of iterations) takes five to seven times longer.

7 Summary

This paper presented an application of ANS for feature extraction, clustering, and classification of tactile impressions in the robotics domain, and revealed some basic characteristics of the BEP network to geometry-, size-, and activation-constrained grey-scale data. Fundamental results indicate that with appropriate network architecture and training sets, ANS are able to extract subtle feature differences and construct high-dimension decision surfaces. These results however, were derived from a very small training sample, and further analysis must be completed before general conclusions can be reached. The response of ANS to noisy data, and their effectiveness in classifying un-trained patterns are being examined. Also, scaling studies should indicate if these concepts can be extended to (at least low-resolution) vision data.

References

[Cottrell *et al.*, 1987] G Cottrell, P Munro, and D zipser. Image compression by back propagation: An example of extensional programming. *ICS Report 8702*, Feb 1987.

[Goddard *et al.*, 1989] N Goddard, K Lynne, T Mintz, and L Bukys. Rochester connectionist simulator. Technical Report 233, University of Rochester, Oct 1989.

[Hering *et al.*, 1990] D Hering, P Khosia, and B Kumar. The use of modular neural networks in tactile sensing. *International Joint Conference on Neural Networks*, 2:355–358, Jan 1990.

[Hillis, 1982] D Hillis. A high resolution imaging touch sensor. *International Journal of Robotic Research*, 1(2):33–44, 1982.

[Kadonoff, 1983] M Kadonoff. 3-d object recognition and orientation with gray-scale data using ternary tree structure and countour centers. Master's thesis, Duke University, Dec 1983.

[Kinoshita, 1973] G Kinoshita. Pattern recognition of the grasped object by the artificial hand. *Proceedings of the 3rd International Joint Conference on AI*, pages 665–669, 1973.

[Kuczewski *et al.*, 1987] R Kuczewski, M Myers, and W Crawford. Exploration of backward error propagation as a self-organizational structure. *IEEE International on Neural Networks*, 2:89–95, 1987.

[Marik, 1981] V Marik. Algorithms of the complex tactile information processing. *Proceedings of International Joint Conference on Artificial Intelligence*, pages 773–774, Aug. 1981.

[Muthurkishnan *et al.*, 1987] C Muthurkishnan, D Smith, D Myers, and J Rebman. Edge detection in tactile images. *Proceedings of IEEE International Conference on Robotics and Automation*, 1987.

[Nicholls and Lee, 1989] H Nicholls and M Lee. A survey of robot tactile sensing technology. *The International Journal of Robotics Research*, 8(3).3–30, June 1989.

[Rebman and Trull, 1983] J Rebman and M Trull. A robust tactile sensor for robot applications. Technical Report LL-2142, Lord Corporation, 1983.

[Rumelhart *et al.*, 1986] D Rumelhart, J McClelland, and PDP Research Group. *Parallel Distributed Processing*, volume v1. MIT Press, 1986.

[Sato *et al.*, 1977] N Sato, W Heginbotham, and A Pugh. A method for three dimensional part identification by tactile transducer. *Proceedings of the 7th International Symposium on Industrial Robots*, pages 577–585, 1977.

[Saund, 1986] E Saund. Abstraction and representation of continuous variables in connectionist networks. *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 638–644, 1986.

[Sietsma and Dow, 1988] J Sietsma and R Dow. Neural net pruning - why and how. *IEEE International Conference on Neural Networks*, 1:325–333, 1988.

[Takeda, 1974] S Takeda. Study of artificial tactile sensors for shape recognition – algorithm for tactile data input –. *Proceedings of the International Symposium on Industrial Robots*, 4:199–208, 1974.

[Togai, 1982] M Togai. Pattern recognition schemes for a touch-sensor array based on moment invariant and circle search methods. Technical Report EE-82-03, Duke University, Aug 1982.

EXPLANATION-BASED LEARNING

Generalizing the Order of Goals as an Approach to Generalizing Number

Henrik Boström
SYSLAB
Department of Computer and Systems Sciences
Stockholm University
Electrum 230, 164 40 Kista
Sweden

Abstract

The common idea among previous approaches for generalizing number is to look for repeated applications of rules (or operators) while generalizing an example proof to produce a general schema. We describe another approach, the algorithm N, that generalizes number from planning problems which are stated in the STRIPS-formalism. Instead of generalizing the structure of an example proof in terms of operator applications, the algorithm generalizes the order in which the literals in the goal state description are reached, yielding a generalized precedence graph. The algorithm N is compared to one of the previous approaches for generalizing number, that can be applied to planning problems which are stated in the STRIPS-formalism. Experiments have shown that schemata produced by algorithm N can be more efficiently utilized than schemata produced by the previous algorithm. Also, the algorithm N is shown to be able to handle a class of problems that the previous algorithm cannot.

1 Introduction

To generalize the structure of an example proof such that a fixed number of rule applications in the proof is generalized into an unbounded number of applications is commonly referred to as *generalizing number*. Another name for the same principle is *generalizing to N*. In this work, we use the term *generalizing number* in a broader sense than restricting the generalizations to be made in terms of rule or operator applications. By generalizing number, we mean the principle of generalizing the solution of a specific example problem into a general schema, that can be used to solve a class of problems, where the members not only may differ from each other with respect to the name of the objects involved, but they may also differ in the number of objects involved. For example, an instance of generalizing number in the Block's World is to learn a general schema for building towers of arbitrary height from a proof of building a tower of four blocks height. Different approaches of generalizing number have been presented in [Shavlik & DeJong 87], [Cohen et al 88], [Cohen 88], [Shavlik 88] and [Shavlik 89].

The common idea among these approaches, while generalizing an example proof to produce a general schema, is to look for repeated applications of rules (or operators). When such a repetition is found, a loop construct is added to the corresponding general schema. We have focused on generalizing number in systems that solve planning problems based on the STRIPS-formalism [Nilsson 82]. The previous approaches to generalizing number have only been considered with generalizations of proofs made by a Horn clause theorem prover. However, one of the previous algorithms [Cohen 88] can also be used to generalize proofs in the STRIPS-formalism. This method has one major shortcoming. The general schema learned by the method, contain information about what operators to select only, and not how to apply them (knowledge about selecting the appropriate *match substitution* according to the terminology in [Nilsson 82]). In the tower building example, a system using a general schema learned by the method to build a certain tower, knows what operator to use in a particular situation (stack, pickup, etc.), but does not know what blocks to involve in the action. This observation has recently been presented in [Boström 89]. In many domains the search space may not be pruned enough by the general schemata learned, to enable new problems of the same class to be solved within acceptable time.

In this paper we present the algorithm N, a new approach to generalizing number in planning problems based on the STRIPS-formalism.

Instead of generalizing the structure of an example proof in terms of operator applications, the algorithm N generalizes the order in which the literals in the goal state description are reached. The algorithm N is described in the next section. In section 3 we present an algorithm for interpreting schemata produced by algorithm N, when solving new problems. A short presentation of how Cohen's algorithm can be applied to planning problems in the STRIPS-formalism is given in section 4. Some preliminary results comparing the algorithm N to the algorithm of Cohen are presented in section 5.

2 The Algorithm N

We first define the input to algorithm N, a precedence graph, that represents all solutions (with some restrictions) to an example problem. Then we describe the algorithm N itself, which produces a generalized precedence graph. Finally, we give some examples of the limitations of the general heuristic used in algorithm N.

2.1 Goal orders and precedence graphs

A *goal order* for a problem, is a sequence of the literals in the goal state description, such that the literals can be added sequentially by applying STRIPS-operators without ever deleting a previously added literal in the goal state description. A *precedence graph* represents all possible goal orders for a certain problem. Below, we give definitions of these two concepts. Examples of precedence graphs for building a tower and an arch are also presented.

Definition: Given an initial state description I_d, a goal state description G_d and a set of STRIPS-operators S. A *goal order* is a sequence $(l_1, ..., l_n)$ containing all literals in G_d exactly once, such that there exists a solution sequence[1] $(s_1, ..., s_m)$ with the following properties:
i) for each literal l_j in $(l_1, ..., l_n)$ there does not exist more than one instance s_i in $(s_1, ..., s_m)$ that contains l_j in its add list (each literal must not be added more than once)
ii) for each pair of literals (l_i, l_j) in $(l_1, ..., l_n)$ such that $i<j$, for each pair of instances (s_k, s_l) in $(s_1, ..., s_m)$, if s_k contains l_i in its add list and s_l contains l_j in its add list, then $k<l$. (each literal preceding another literal in the sequence must be added before the other)

Definition: Given an initial state description I_d, a goal state description G_d and a set of STRIPS-operators S. A *precedence graph* G is a directed acyclic graph, where each node is labeled with a literal, and that has the following properties:
i) every literal in G_d appears exactly once in the graph, and there does not exist a literal in G that is not member of G_d (all literals in the goal state description, and only those, must be mentioned in the graph)
ii) there exists a subset g of the nodes in G such that each node in the graph that is not member of g can be reached by following the arcs from a node in g, and there exists a node in the graph that can be reached from each node in g (all nodes in the graph must be connected)
iii) there does not exist a goal order $(l_1, ..., l_n)$ with respect to I_d, G_d and S, such that there exists a pair of literals (l_i, l_j) in the sequence where $i<j$ and l_i can be reached from l_j in G by following the arcs (a literal that precedes another literal in a goal order must not be reachable from the other literal in the graph)
iv) there do not exist three different literals a, b and c in G such that b and c are successors of a, and c can be reached from b by following the arcs, and there does not exist a pair of literals (a, b) such that there are more than one arc from a to b (redundant arcs are not allowed).

In an example proof one particular goal order is easily found, by regarding the order in which the literals in the goal description are added. But in order to find the corresponding precedence graph to the example problem we need all goal orders for the particular problem, i.e. all possible ways of solving the problem without deleting a previously added literal in the goal state description. An algorithm for deriving a precedence graph, given all goal orders for a problem is presented in [Boström 90]. Henceforth, we assume that the corresponding precedence graph is given for each problem, instead of all goal orders.

Example 1
Given the initial state description I_d = {handempty, ontable(a), clear(a), ontable(b), clear(b), ontable(c), clear(c), ontable(d), clear(d)}, the goal state description of a tower G_d = {on(a, b), on(b, c), on(c, d)} and the set of

[1] A solution sequence is a sequence of instances of STRIPS-operators, such that when applied to the initial state description, a state description is produced from which the goal state description logically follows.

STRIPS-operators for Block's World [Nilsson 82, p281]. The precedence graph (and the corresponding tower) is shown in figure 1. The precedence graph is linear[1] since there exists exactly one goal order, namely (on(c, d), on(b, c), on(a, b)).

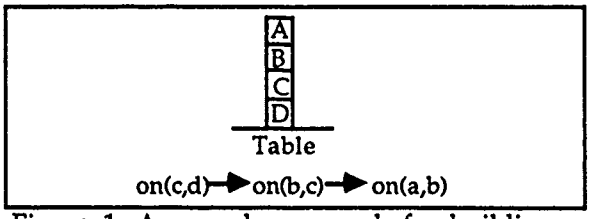


Figure 1. A precedence graph for building a tower.

Example 2

Assume that we extend the set of STRIPS-operators in example 1, with a stack2(X, Y, Z) operator that can be used to stack a block on two blocks. Given the initial state description where all blocks are on the table, the goal state description that corresponds to the arch in figure 2 and the extended set of STRIPS-operators. A precedence graph for building the arch is shown in figure 2 (below the arch). The precedence graph in figure 2 is not linear since there exist more than one goal order for the problem.

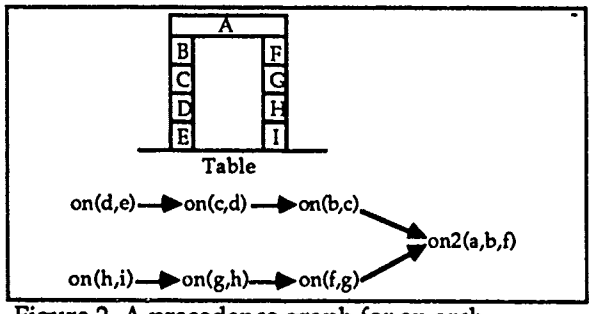


Figure 2. A precedence graph for an arch.

2.2 Generalizing Precedence Graphs

The principle of generalizing number was defined as the principle of generalizing the number of objects involved in a specific example in order to produce a general schema. Generalizing number from examples of building a tower or an arch means producing general schemata that can be used to solve problems of building towers and archs involving an arbitrary number of objects. The commonly adopted heuristic in algorithms for generalizing number is to look for some kind of repetition in a proof of an example problem. In contrary to previous approaches for generalizing number, we do not look for repetition of operator applications in a proof. Instead, we look for repetition in the corresponding precedence graph to the example problem. The question is: repetition of what?

The purpose of generalizing a precedence graph is to be able, when solving a problem, to give a partial ordering of the literals in the goal state description. This ordering can then be used to produce a solution sequence. A precedence graph can be viewed as an instance of a schema that declares order constraints on the literals in the goal description. We want our algorithm to determine these order constraints by regarding the precedence graph that corresponds to the example problem. Then the heuristic of looking for repetition of order constraints in the precedence graph can be used in order to generalize number.

We assume that the order constraints in a precedence graph can be expressed as relations between literals that are directly connected by arcs. The algorithm must determine for each pair of literals (a, b) in the precedence graph, such that b is a successor of a, how they are related. The following heuristic is used in algorithm N: "two literals can be said to be related if they share at least one object constant or if they have different object constants that appear as arguments of the same literal in the initial state description". Some limitations of this heuristic are presented in section 2.3.

The generalization of a precedence graph is made by the algorithm N in two steps. First, a general version of the precedence graph is produced where the arcs represent relations between the literals in the precedence graph. Then the new graph is reduced, by merging nodes having arcs with common relations, yielding the final generalized precedence graph. Below, we give an overview of the algorithm N. A detailed description of the algorithm is found in [Boström 90].

[1]By a linear precedence graph, we mean a precedence graph where each node does not have more than one successor and each node does not have more than one predecessor.

Algorithm N
Input: a precedence graph *G* and an initial state description I_d
Output: a generalized precedence graph *G'*

1. Let *G'* be a graph consisting of the nodes in *G*. For each pair of nodes ($n1$, $n2$), such that $n2$ is a successor of $n1$ in G, add an arc from $n1$ to $n2$ to G'.
   Label all arcs with triples, where the first and second argument are the corresponding literals to $n1$ and $n2$, with each constant replaced with a unique variable. The third argument is a set of relations describing the relation between the two literals, according to the heuristic that literals can be related by sharing constants, or by having different constants as arguments that appear in the same literal in the initial state description.

2. Merge all pair of nodes ($n1$, $n2$) where $n2$ can be reached from $n1$, and where both nodes have arcs leading from them labeled with the same triples.

Example 1 cont.
Application of step 1 in the algorithm to the precedence graph in figure 1 results in the graph shown in figure 3a. The only relation found between the literals 'on(c, d)' and 'on(b, c)' in the precedence graph, is that equality holds between the first literal's first argument and the second literal's second argument. There exists no literal in the initial state description that has more than one argument, and thus there is no literal that contains two different constants from the literals. The same relation between the literals 'on(b, c)' and 'on(a, b)' in the precedence graph is found by step 1 in the algorithm.

After step 2, the graph in figure 3a is generalized into the generalized precedence graph in figure 3b. Two nodes have been merged, since one of them could be reached from the other, and from both nodes arcs were leading labeled with the same triples.

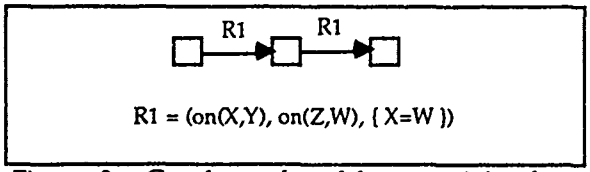


Figure 3a. Graph produced by step 1 in algorithm N for the tower building example.

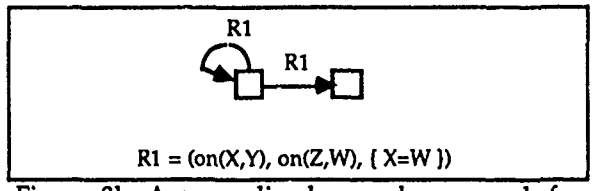


Figure 3b. A generalized precedence graph for building towers.

Example 2 cont.
Application of step 1 in the algorithm N to the precedence graph in figure 2 results in the graph shown in figure 4a. As we can see, the same relation has been found as in the previous example (denoted R1), but also two other relations have been found.

In step 2, two pairs of nodes are merged, yielding the generalized precedence graph in figure 4b. No more nodes in the resulting generalized precedence graph allow merging, since there does not exist a pair of nodes from which identically labeled arcs lead, and where one node can be reached from the other.

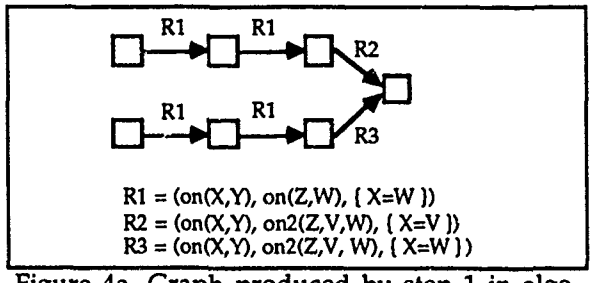


Figure 4a. Graph produced by step 1 in algorithm N for the arch building example.

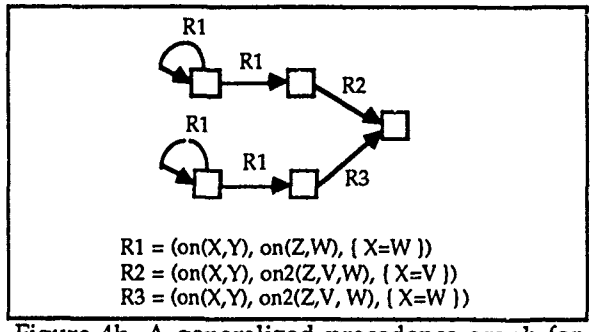


Figure 4b. A generalized precedence graph for building archs.

2.3 Limitations of the Heuristic in Algorithm N

Three types of incorrect generalizations can be noticed due to the heuristic used in the algorithm N:

i) over-generalization as a consequence of the algorithm's unability to find a relation between two literals in the precedence graph (i.e. there does not exist a literal in the initial state

description that contains object constants from both of the literals and the literals have no object constant in common)
ii) under-generalization as a consequence of the algorithm's consideration of a literal in the initial state description that is irrelevant, when looking for a relation between two literals in the precedence graph
iii) under-generalization as a consequence of an assumption by the algorithm that two literals in the precedence graph have a common object constant by necessity, when it is by a coincidence.

3 Interpretation of Generalized Precedence Graphs

A generalized precedence graph is used to give a partial ordering of the literals in the goal state description, that can be used to produce a solution sequence. In this section we describe the algorithm *Find-solution* that can be used to find a solution sequence for a particular planning problem, given a generalized precedence graph.

The algorithm works in three steps. First, a set of tree structured graphs are produced constraining the order of the literals in the goal state description. Second, a potential goal order is derived using the set of trees. Third, a domain independent means-end analysis system is used to produce a solution sequence, given the potential goal order.

The algorithm can be summarized as follows. For algorithmic details, see [Boström 90].

Algorithm Find-solution
Input: a initial state description I_d, a goal state description G_d, a set of STRIPS-operators O and a generalized precedence graph G'
Output: a solution sequence S

1. A set of tree structured graphs is created in the following way. First, literals in the goal state description are matched to the second argument of relations that label arcs leading to nodes without successors. For each literal that matches such an argument, a tree is constructed where the literal is the root. From each root, there may lead branches to other literals. These literals are such that they match the first argument of a relation that labels an arc, that leads to the node of the root in the generalized precedence graph. In addition, all facts in the third argument must be true with respect to the initial state description and with respect to the equivalence relation. From the literals that match the first argument of the relation, trees are created in a recursive manner starting at the nodes from which the arcs lead in the generalized precedence graph.

2. A potential goal order is produced, given the set of tree shaped graphs, in the following way. The goal order is initialized to the empty sequence. Iteratively, it is checked if there exists a literal that is not member of the sequence, such that each literal from which it can be reached in any of the graphs, is a member of the sequence. If that is the case, then the literal can be added to the end of the sequence.

3. Given a goal order, a solution sequence is produced using a means-ends analysis technique [Nilsson 82, p304]. The main difference between our algorithm and the original algorithm is that our algorithm does not have to non-deterministically select a literal in the goal, and that no domain knowledge is used to select an operator to reduce the difference between the current state description and the goal state description. The algorithm also checks that a previously added literal in the goal description is never deleted.

Example 1 cont.
The goal is to build the tower in figure 5a with use of the the original set of STRIPS-operators from the initial state where all blocks are clear and on the table. Given the generalized precedence graph in figure 3b. All literals in the goal state description can match the second argument of the triple '(on(X, Y), on(Z, W), {X=W})'. Hence, when applying step 1 in *Find-solution*, a tree is constructed for each literal in the goal state description, with the literal as a root. The set of tree structured graphs are shown in figure 5b. The tree consists of a single literal, when the literal 'on(e, f)' is the root, since there does not exist another literal in the goal state description that matches the first argument, such that the equality holds.

Application of step 2 to the set of tree structured graphs produces the goal order (on(e, f), on(d, e), on(c, d), on(b, c), on(a, b)), since it is the only possible goal order. The solution sequence produced by step 3 is (pickup(e), stack(e, f), pick(d), stack(d, e), pickup(c), stack(c, d), pickup(b), stack(b, c), pickup(a), stack(a, b)).

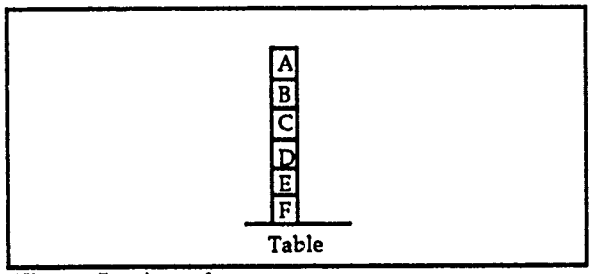


Figure 5a. A goal tower.

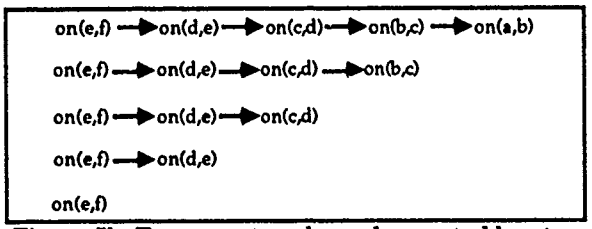


Figure 5b. Tree structured graphs created by step 1 in *Find-solution* for the tower building problem.

4 Cohen's Algorithm

We briefly present the algorithm of Cohen [Cohen 88]. The algorithm is intended to be applied to proofs made by a Horn clause theorem prover. However, in this section we show how the algorithm also can be applied to proofs given in the STRIPS-formalism.

Cohen's algorithm learns deterministic control automata from example proofs. Each arc of a control automaton is labeled with an input symbol representing a set of rules, and an output symbol representing a single rule. The control automaton guides a proof as follows: first all applicable rules are collected into a set S. If there is an arc leading from the current state, with input symbol S, then the rule corresponding to the output symbol is applied. The new current state will be the one the arc was leading to. If there is no arc with input symbol S, the proof fails. The proof succeeds if it is completed using the sequence of rules output by the control automaton.

Cohen's algorithm applied to a proof of building the tower in figure 1 produces the initial control automaton in figure 6a. The next step in the algorithm checks the possibility of merging states having arcs with the same input/output symbols (if it is possible to make the automaton deterministic after merging). All possible mergings are made, yielding a reduced deterministic control automaton. The corresponding reduced automaton to the initial control automaton, is presented in figure 6b.

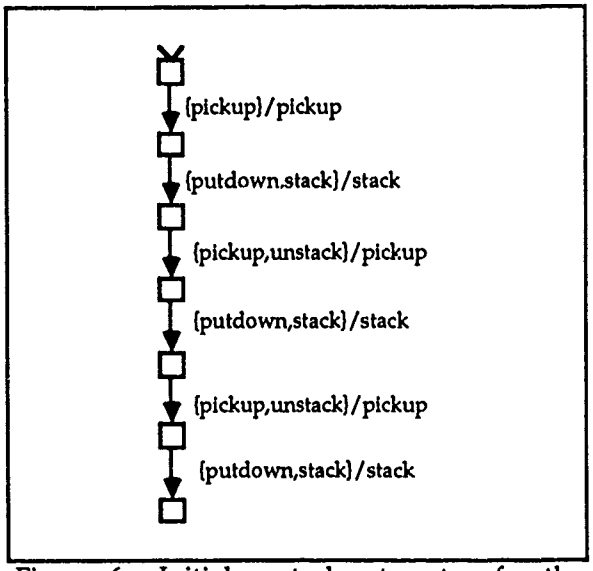


Figure 6a. Initial control automaton for the tower building example.

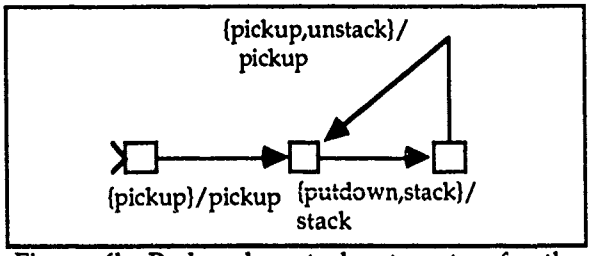


Figure 6b. Reduced control automaton for the tower building example.

In Cohen's system ADEPT, backtracking is not permitted. In the case, when the system has learned to build towers, and a new tower is to be built, the system will not succeed if the wrong block is first picked up. Since the probability for incorrectly choosing a block is very large, the learned procedure will seldom (or in more complex cases never) work. However, in our comparison we permit the system to backtrack when a dead-end is reached.

5 Results

In this section we present some preliminary results from the comparison of the algorithm N with the algorithm of Cohen. Two main questions are raised in this section:

a) Which of the two algorithms produces the most efficient general schema?

b) Are there any examples that one of the algorithms does learn from, but the other does not?

The efficiency of the learning algorithms is not considered in this comparison.

The size of the search tree when using a schema produced by Cohen's algorithm, is entirely dependent on the number of possible opera-

tor applications. When a generalized precedence graph is used, the search tree depends on the number of literals that matches relations in the generalized precedence graph and on the search made by the domain independent means-ends analysis algorithm given the goal order.

Since these numbers may vary between domains and problems it is difficult to give a general answer. However, we have made a number of limited experiments comparing the efficiency of the general schemata produced by the algorithm N and the algorithm of Cohen. Figure 7 shows the growth of time taken to build towers of different height with schemata produced by Cohen's algorithm and the algorithm N. In addition, we present the performance of a domain independent means-ends analysis system, that in contrast to the means-ends analysis system used when interpreting generalized precedence graphs, has to non-deterministically select a literal in the goal state description. All problems given to the systems are stated as worst-case problems (i.e. the literals in the initial state description and goal state description are ordered so that the appropriate literals are chosen last when instantiating operators and selecting subgoals respectively). The algorithms are implemented in PROLOG on a Macintosh II. The learning case is the problem of building a tower of 4 blocks.

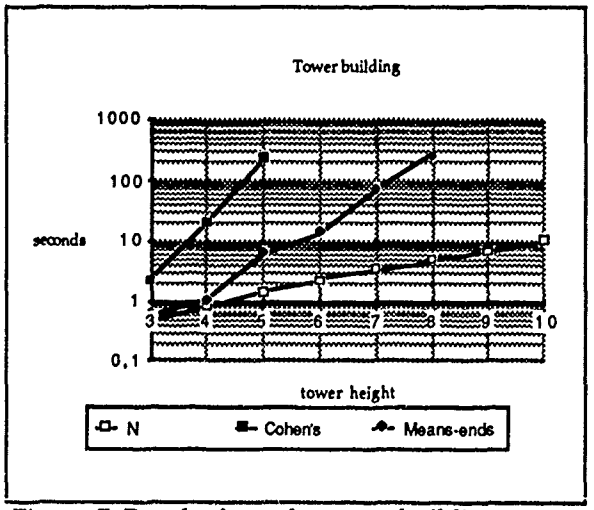


Figure 7. Results from the tower building experiment.

Two examples can be given as an answer to the second question stated above. Cohen's algorithm is able to learn the control automaton in figure 6b from a proof of building a tower with three blocks. The algorithm N needs at least three literals in the goal state description to be able to generalize, and thus cannot generalize a proof of building a tower with three blocks (two literals). On the other hand, there are many classes of problems that the algorithm N can learn schemata for and the algorithm of Cohen cannot. In figure 8 we show an initial control automaton for building an arch, that cannot be reduced and made deterministic. The algorithm N is able to generalize the corresponding precedence graph (as was shown in example 2) and thus can learn from an example problem that Cohen's algorithm cannot.

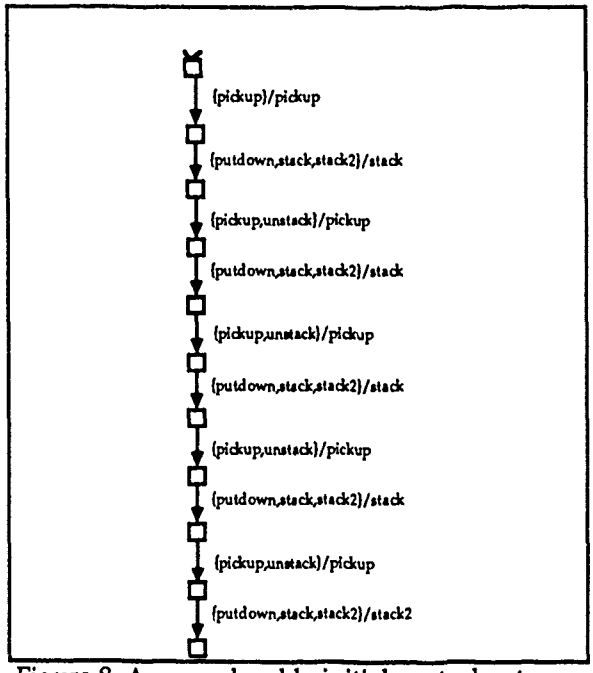


Figure 8. An unreducable initial control automaton.

6 Conclusions

We have presented algorithm N, a new approach to generalizing number that can be used for planning problems which are stated in the STRIPS-formalism. Instead of generalizing a single proof in terms of operator applications, our algorithm generalizes all solutions to the example problem, by generalizing the order in which the literals in the goal state description can be reached, yielding a generalized precedence graph.

The algorithm N is compared to an algorithm for generalizing number that generalizes proofs in terms of rule applications [Cohen 88]. Experiments have shown that it takes less time to find a solution sequence using a generalized precedence graph than finding a proof with a control automaton in a number of cases.

The algorithm N is shown to be able to handle a class of problems that the algorithm of Cohen

cannot. These problems are such that it is not possible to make deterministic reduced automata that correspond to the example proofs. On the other hand, we have shown how Cohen's algorithm can learn from a smaller example than the algorithm N.

The algorithm N can be improved with respect to a number of aspects. The general heuristic used in the algorithm for finding repetition in a precedence graph, may be revised, since there exists problems that cannot be generalized appropriately (see section 2.3).

Another direction to proceed in, is to develop an algorithm that can combine schemata learned from different examples when solving a problem. Up till now, new problems have been solved using a single schema only. Another question is how to use the generalized precedence graphs to solve problems that are subproblems of the problems that the graphs represent.

Acknowledgements

I would like to thank Carl-Gustaf Jansson, Peter Idestam-Almquist and all other members of the ACTA project for good suggestions, and Steven Minton for giving valuable references. Thanks also to the anonymous reviewers for helpful comments.

References

[Boström 89]
Boström H., "Improving Generalizing Number with Conflict Resolution", SYSLAB WP 162, Stockholm (1989)

[Boström 90]
Boström H., "Generalizing Goal Orders as an Approach to Generalizing Number", Licentiate thesis, Department of Computer and Systems Sciences, Stockholm University, Kista, Sweden (in press)

[Cohen et al 88]
Cohen W., Mostow J. and Borgida A.,"Generalizing Number in Explanation-Based Learning", AAAI Spring Symposium on Explanation-Based Learning, Stanford (1988)

[Cohen 88]
Cohen W. W., "Generalizing Number and Learning from Multiple Examples in Explanation-Based Learning" , *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, MI (1988) pp256-269

[Nilsson 82]
Nilsson N. J., *Principles of Artificial Intelligence*, Springer Verlag, Berlin Heidelberg (1982)

[Shavlik 89]
Shavlik J. W.,"Acquiring Recursive and Iterative Concepts with Explanation-Based Learning", Technical Report, Department of Computer Science, University of Wisconsin, Madison, WI (1988)

[Shavlik 88]
Shavlik J. W.,"Issues in Generalizing to N in Explanation-Based Learning", AAAI Spring Symposium on Explanation-Based Learning, Stanford (1988)

[Shavlik&DeJong 87]
Shavlik J. W. and DeJong G. F.,"BAGGER: An EBL System that Extends and Generalizes Explanations", *Proceedings of the National Conference on Artificial Intelligence*, (1987)

Learning Approximate Control Rules Of High Utility

William W. Cohen
Computer Science Department
Rutgers University
New Brunswick, NJ 08903

Abstract

One of the difficult problems in the area of explanation based learning is the *utility problem*; learning too many rules of low utility can lead to *swamping*, or degradation of performance. This paper introduces two new techniques for improving the utility of learned rules. The first technique is to combine EBL with inductive learning techniques to learn a better set of control rules; the second technique is to use these inductive techniques to learn approximate control rules. The two techniques are synthesized in an algorithm called *approximating abductive explanation based learning* (AxA-EBL). AxA-EBL is shown to improve substantially over standard EBL in several domains.

1 Introduction

One of the difficult problems in the area of explanation based learning is the *utility problem*. The *utility* of a rule is its contribution to performance improvement; the utility is directly proportional to the *coverage* of a rule and inversely proportional to the *match cost* of a rule, where *coverage* is defined as the percentage of the time that the rule is used in problem solving, and *match cost* is simply the time needed to determine if a rule is applicable.

The utility problem often arises in learning rules intended to improve the performance of a problem solver. Learning too many rules of low utility can lead to *swamping*, or degradation of performance: in the worst case, a problem solver can be slower after learning than before learning. Previous research on the utility problem has focused on detecting and discarding rules of low utility [Minton, 1988; Markovitch and Scott, 1989], lowering the match cost of rules using partial evaluation or other simplification techniques [Prieditis and Mostow, 1987; Minton, 1988; Tambe and Rosenbloom, 1989], and constraining the use of learned rules [Mooney, 1989].

This paper introduces two new techniques for improving the utility of learned rules. The first technique is to combine EBL with *inductive learning techniques* to learn a better set of control rules. The second technique is to use these inductive techniques to learn *approximate control rules*: that is, to learn rules which are approximations of the control rules which would be learned by standard EBL techniques. Although use of inductive techniques alone does not always lead to rules of higher utility, the combination of these techniques is shown to substantially improve the performance of EBL across a variety of domains.

Combining EBL with inductive learning. Flann and Dietterich have noted that in the process of learning control rules, many EBL systems make inductive leaps. An example given in [Flann and Dietterich, 1989] is a rule learned by LEX2 in the symbolic integration domain:

> If the current problem matches
> $\int cx^r dx (r \neq -1)$
> Then use the operator
> $\int cf(x)dx \Rightarrow c \int f(x)dx$

Forming this rule is an inductive leap; this is witnessed by the fact that the rule recommends the *wrong* action on the problem $\int 0x^4 dx$ (multipying out the zero to get $\int 0dx$ leads to a shorter solution). Another case in which inductive decisions are made is when only one of several possible control rules is learned; for example, PRODIGY might learn either a macro-operator or a set of operator preference rules from a trace [Minton, 1988].

However, the standard implementation of EBL, in which control rules are learned incrementally as needed, is relatively slow as an inductive learner. Combining EBL with more powerful inductive learning techniques can improve the rate at which learning converges to an adequate set of control rules. A crucial point is that on average, *improving the rate of convergence will also improve the coverage of the individual rules learned.* This is true because in order to improve the learning rate, it necessary to find *fewer* rules with *better* coverage.[1]

[1] It must be emphasized that we are *not* advocating use of pure inductive learning techniques (*e.g.*, version spaces) to learn control rules. Explanation-based techniques are clearly more appropriate in learning control rules because there is a strong theory.

Using approximate rules. The first technique alone, however, is not enough to avoid swamping. The problem is that in learning control rules, one is trying to maximize the utility of the set of rules. To do this, one must simultaneously *maximize total coverage*, and *minimize total match cost*. Standard inductive learning techniques, however, maximize coverage without regard to match cost.

Viewed in this light, *learning useful control rules is a multiple-resource optimization problem*, in which the two antagonistic resources of coverage and match cost must be simultaneously optimized. A standard technique for solving such problems is to set an artificial constraint on one resource, and optimize the other resource subject to this constraint. This technique can be applied to the control rule learning problem by constraining the inductive learner to only consider control rules with match cost below a certain fixed cutoff.

Unfortunately, the control rules learned by explanation based techniques are almost always very complex; hence if a reasonable match-cost cutoff were chosen, very few rules would be available for the inductive learning method to consider. This problem can be remedied by allowing the learner to also consider *approximations* to expensive control rules.

In the remainder of the paper, we first discuss the control rule learning problem used to test these ideas, and then the learning algorithm used. We then describe the domains used for experimentation. Finally, we present and interpret our experimental results and draw some conclusions.

2 The learning problem

To evaluate the learning algorithms discussed in this paper, we used as a testbed the problem of learning *clause selection rules* for Prolog programs. This learning problem has several advantages.

- Commercial Prologs are highly optimized, and hence the initial problem solver has a fairly low overhead. This encourages careful experimentation, and also means that comparisons of performance before and after learning are a reasonable test of the overall effectiveness of learning.
- Many different problem solving strategies (for example, state space search, problem decomposition, means-end analysis, etc.) can be easily coded as Prolog programs in such a way that learning clause selection rules significantly improves performance.

Our learning testbed consists of two routines. The first is a *critic* which analyzes a set of Prolog traces and extracts examples of correct and incorrect clause selection decisions. A *correct* decision is any decision which leads to a solution; an *incorrect* decision is any decision which is eventually retracted by Prolog's backtracking strategy. Along with each correct decision, a proof of its correctness is recorded. The theory used to derive this proof is very simple: it merely contains the clauses of the initial program as axioms, and also an axiom of the form

$$\text{useful}(c_i, A) \leftarrow B_1, \ldots, B_q$$

for each clause $c_i = (A \leftarrow B_1, \ldots, B_q)$ in the initial program. This 'usefulness theory" is an extension to the problem of Prolog clause selection of the theory used in the LEX/2 system [Mitchell, 1982] for operator selection.

The second routine is a *control rule compiler* which, given a set of control rules and an initial program, generates a new Prolog program which incorporates the learned control rules. The routine takes as input a set of control rules of the form

$$\text{useful}(c_i, A) \leftarrow \mathcal{U}(A)$$

which can be read as saying "if $\mathcal{U}(A)$ is true, then it is useful to apply clause c_i to the goal A". $\mathcal{U}(A)$ is typically a complex conjunctive condition involving the variables of the goal A. For each such rule, the post-processor constructs a copy of clause c_i to which $\mathcal{U}(A)$ has been added as an additional "filtering condition"; in other words, if $c_i = (A \leftarrow B_1, \ldots, B_q)$, then a clause of the form

$$A \leftarrow \mathcal{U}(A), !, B_1, \ldots, B_q$$

is constructed. Notice that the cut (!) makes selection of this clause a *committed choice*: no backtracking will be done if this choice is incorrect.

One advantage of incorporating control rules in this way is that matching is done by Prolog's unification procedure; directly using the implementation language reduces the overhead of pattern matching.

If there is more than one control rule for c_i, then multiple copies of c_i are generated: *i.e.*, multiple control rules are interpreted disjunctively. A copy of the original clause with no filtering condition is retained if and only if some decisions to use that clause are not explained by any control rule. Clauses in the new program are ordered first according to the order of the clauses in the initial program to which they correspond (since this ordering may represent important control information), and then according to the ordering of control rules.

No simplification or compression is done.

Retaining the original clauses when some control decisions are unexplained means that some performance improvement can be gained even if control rules can only be learned for some clause selection decisions. This makes the learning system less brittle; however, it also means that some backtracking may still be done in the learned program. This is the motive for discarding the original clauses when all control decisions have been explained, and requiring that learned control decisions are not backtracked over; otherwise, in backtracking, all of the solutions considered by the original program would still have to be examined and rejected, limiting the degree to which performance can be improved. However, discarding clauses also means that the learned program may fail on problems which are solvable by the initial program. When this happens the original program is invoked on the top-level goal. Thus, in spite of using approximate control rules and

committed choice control decisions, the learned program always solves at least as many problems as the original program did.

3 The learning algorithms

Three learning algorithms were experimentally compared. The first two are strawmen, corresponding to a conventional EBL control-rule learning system, and to an extended EBL learning system which uses inductive learning techniques to improve coverage, but which ignores match cost. The third algorithm uses inductive techniques to search a space of approximate rules, and is the main focus of this research.

One important difference between these algorithms and conventional control-rule learning algorithms is that they are *non-incremental*. Each learning algorithm takes as input a set of examples of correct and incorrect control decisions, which are generated by applying the critic to a large set of traces.

Standard EBL (EBL): Standard EBL starts with an empty list of control rules, and examines each correct control decision in the order in which they were made. If there are no existing control rules which account for the decision, then EBG is applied to the proof of the correctness of the control decision, and the resulting rule is added to the end of the list of learned control rules.[2]

Standard EBL is a batch simulation of a SOAR-like incremental learning system which uses a learned clause selection rule if one is available, and which otherwise first uses search to determine the right clause, then forms a clause selection rule which summarizes the results of the search process.

Abductive EBL (A-EBL): Abductive EBL also starts with an empty list of control rules, but repeatedly adds to the end of the list the control rule R which a) is *consistent*, *i.e.*, does not explain any incorrect control decisions, and b) maximizes the ratio of the number of unexplained decisions explained by R to the size of R.[3] R is chosen from a *candidate pool* of rules which consists of all rules which could be generated by applying EBG to some correct control decision.

The main loop in the A-EBL algorithm implements a greedy set cover of the control decisions; in other words, A-EBL looks for a minimal-size set of control rules to explain all the control decisions. The greedy set cover technique has also been used in Haussler's algorithm for learning disjunctive boolean formulae. A-EBL has been used with some success on inductive learning tasks [Cohen, 1989; Cohen, 1990]. Its main advantage over standard EBL techniques for such tasks is that it can be used even on an "abductive" domain theory — one which generates multiple inconsistent explanations. A-EBL has been shown to satisfy Valiant's criterion of pac-learnability, and to have near-optimal sample complexity [Cohen, 1989].

Approximating abductive EBL (AxA-EBL): Approximating abductive EBL works exactly like A-EBL, except that the candidate pool consists of all *k-bounded approximations* to a rule generated by applying EBG to some correct control decision. A k-bounded approximation is formed by dropping all but j conditions from the body of the rule, for some $j \leq k$.

A k-bounded approximation rule will have low match cost, unless the operational conditions are expensive to test. For example, if the truth of operational conditions is tested by database lookup in a database of n facts, then the match cost of a k-bounded approximation is $O(n^k)$.

AxA-EBL is more computationally expensive than EBL; it runs in time polynomial in the total size of the proofs of the control decisions, but exponential in k. Hence, only small values of k can be used. In the current implementation, two techniques are used to prune the search space of approximate rules: Prolog mode declarations are used to eliminate ill-formed approximations, and an admissible heuristic search is used to find the optimal rule from the candidate pool. These tricks substantially reduce learning time; the current implementation of AxA-EBL takes an average of 15.5 CPU minutes on a Sun/4 to process 120 traces from the STRIPS robot-world domain described below. (It takes about 20 minutes to solve the 120 problems using the original means-ends analysis planner.)

A-EBL is also computationally expensive, but for a different reason: to find the optimal rule in the candidate pool, each candidate must be tested against every control decision. This is very expensive if rules have a high match cost.

Standard EBL, A-EBL, and AxA-EBL share two properties which appear to be crucial for success on this learning task. First, all three algorithms can be constrained to produce rules which contain only operational features. Second, like standard EBL, both A-EBL and AxA-EBL usually learn *several* control rules which determine when a particular clause should be selected: in other words, the concept of "usefulness" for each clause can be *disjunctively* defined. The latter property is important because many Prolog programs contain clauses which are useful in several different situations. Usually, several control rules, interpreted disjunctively, will be needed to determine when such a clause is useful.

A-EBL and AxA-EBL differ in these respects from other inductive extensions to EBL, in particular mEBG and IOE [Flann and Dietterich, 1989].

4 Experimental results

4.1 Description of the domains

Experimentation has been done with these three learning algorithms in several domains. The domains are summarized in Table 1.

μLEX: A simplified version of symbolic integration using state-space search with iterative deepening. The training and test problem sets and the operator set are

[2] Experimental studies [Shavlik, 1987] suggest that this ordering is most beneficial.

[3] The size of a rule is defined to be the number of nodes in the explanation structure used to form the rule.

Table 1: Summary of domains used in experiments

Domain	Description	Search Strategy	Number of Operators	Training Set Sizes	Test Set Size
μLEX	simplified LEX	state-space	36	4,8,12,16,19	13
RW	STRIPS robot world	means-ends	10	5,15,25,45,80,120	50
BW	Nilsson blocks world	means-ends	4	5,15,25,45,80,120	100
GRID	graph search	depth-first	3	5,10,15,25	100

taken from [Keller, 1987].

RW: The STRIPS robot world of [Fikes *et al.*, 1972], with a means-ends analysis planner. Problems for the training and test set are generated by selecting a floorplan randomly from the three given in [Minton, 1988], randomly placing 3 blocks and the robot, and then taking a random walk of bounded length in state space.

BW: The blocks world of [Nilsson, 1987], with the identical means-ends analysis planner. Problems are generated by constructing an initial scene with 20 randomly placed blocks, and then taking a random walk of bounded length in state space.

GRID: Depth-bounded depth-first search of an artificial graph. The graph is a 15x15 grid, with 75 randomly placed obstacles, and 5 randomly placed "cities", which are tightly connected by a network of five "interstate highways". There are three kinds of operationally different connections in the graph, corresponding to increasing or decreasing the value of a coordinate by 1 or following an "interstate". Problems are selected by randomly picking start and end points, and asking for a path between the points.

BW and RW problems can be quite difficult, so the planner includes two resource bounds: first, a bound on the maximal length of the plan, and second, a time bound. The time bound was set at 60 seconds for these experiments: almost all of the blocks world problems and about 6/7 of the robot world problems were solvable in this time bound.

The domains are listed roughly in order of their difficulty for standard EBL techniques. For instance, symbolic integration is well suited to use of EBL for operator selection; however, unconstrained use of EBL in the blocks world domain can lead to swamping [Minton, 1988; Mooney, 1989]. Graph searching is a very difficult problem for standard EBL techniques because the match cost of learned rules is very high, and the cost of problem solving without any learned rules is low; matching the rules learned by EBL is equivalent to solving the NP-complete subgraph isomorphism problem [Minton, 1988], whereas depth-first search only requires time linear in the size of the graph.

4.2 Comparison of EBL and AxA-EBL

A series of experiments were designed to determine how the performance of a learned program varied as a function of the number of training examples used in learning. In each experiment, first a *test set*, of the size indicated in table 1, was selected. In the case of BW, RW and GRID, the test problems were randomly selected; in μLEX, the test problems were the designated test problems in [Keller, 1987] (problem sets AT and BT.) Then a *training set* was selected, again randomly for BW, RW and GRID, and from the designated training problems (problems sets A and B) for μLEX. Progressively larger subsets of the training set were then given to each learning system, again as indicated in table 1, and the control rules learned from each subset were applied to the problems in the test set. Two statistics were kept. First, the *total time* required to solve all of the test problems was recorded. The second statistic kept was the *coverage* of the set of control rules: the percentage of the test problems that the learner program solved directly, without recourse to the original program.[4] These experiments were repeated ten times, and the results were averaged. In BW, RW and GRID, each trial used a different randomly selected training set; in μLEX, each trial used a different ordering of the fixed training set.

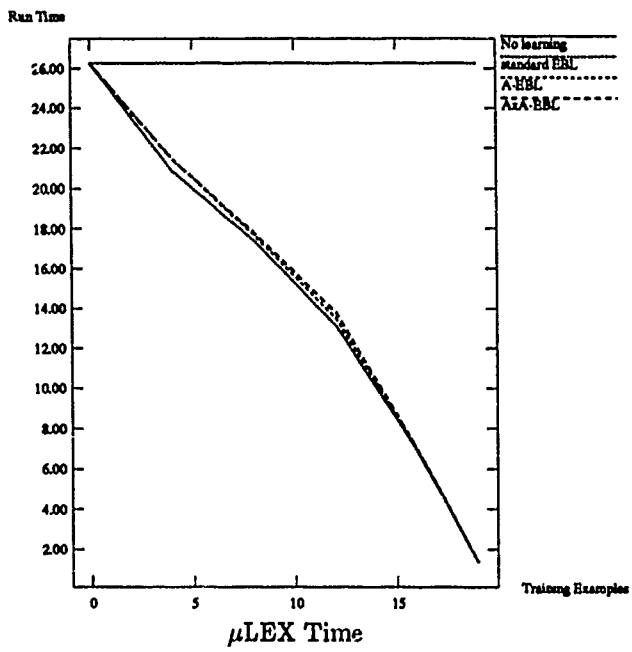
Approximations in the domains were chosen by incrementing k until either some performance improvement occurred, or until the cost of running AxA-EBL became excessive. One of the surprises of these experiments was the expressiveness of a language of very short approximations: in all of the domains except GRID, substantial performance improvement occurred using AxA-EBL with $k \leq 2$.

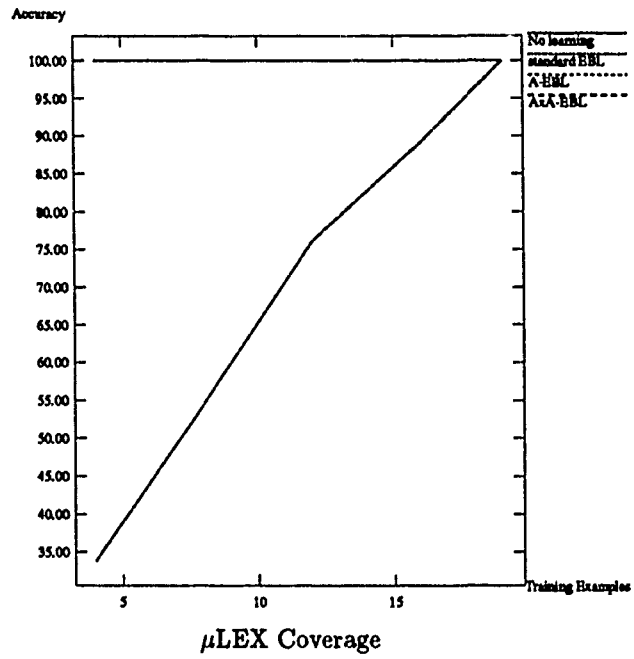
The results of these experiments are summarized in table 2. The crucial column of table 2 is the last one, which shows the ratio of the time spent by the program learned by AxA-EBL in solving the problems of the test set to the time spent by the program learned by EBL. AxA-EBL is marginally slower than EBL on the μLEX domain, and substantially faster on the remaining domains; overall, the programs learned by AxA-EBL are about twice as fast as the programs learned by EBL.

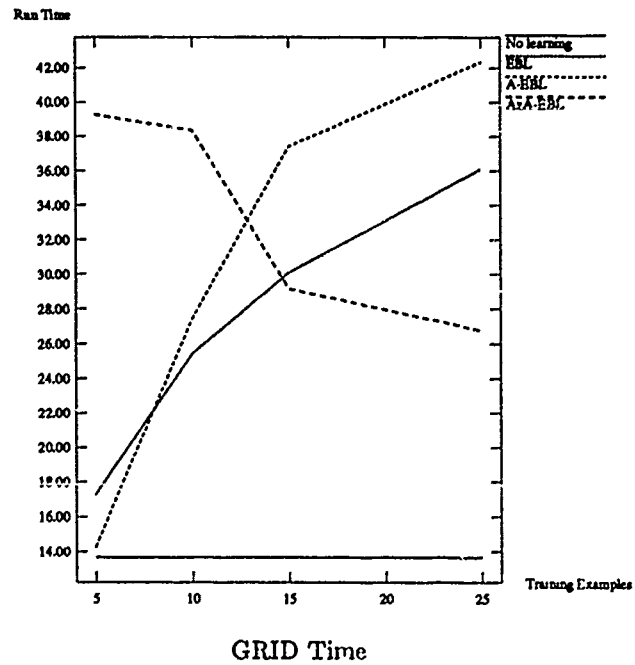
The results of the experiment are given in more detail Figure 1 and Figure 2. These figures graph the average time spent by a learned program, and also the average coverage of the learned program (*i.e.*, the percentage of the time that it was *not* necessary to go back and use the original program to solve a problem) as a function of the number of training examples given.

The three learning techniques behaved essentially

[4] Recall that the program which incorporates the learned control rules may fail if not enough control rules have been learned, or if some of the control rules are incorrect. If the program fails, then the original problem solver is invoked on the problem.







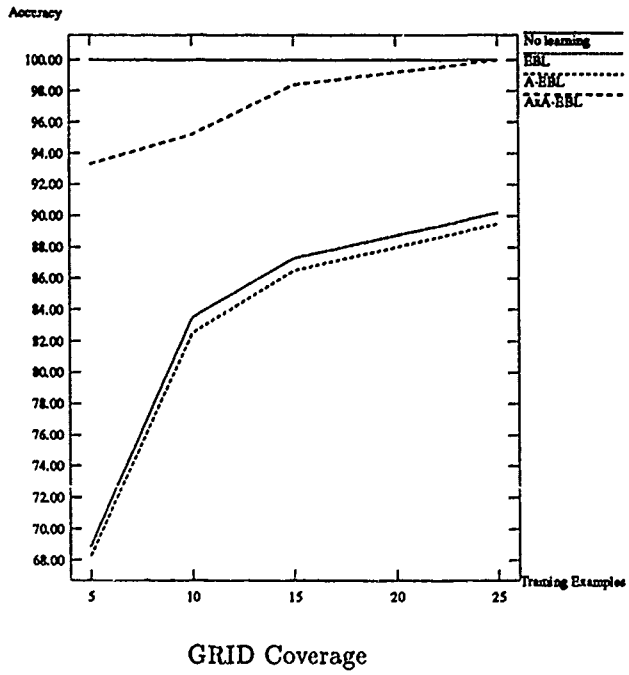


Figure 1: Performance and Coverage Curves for μLEX and GRID

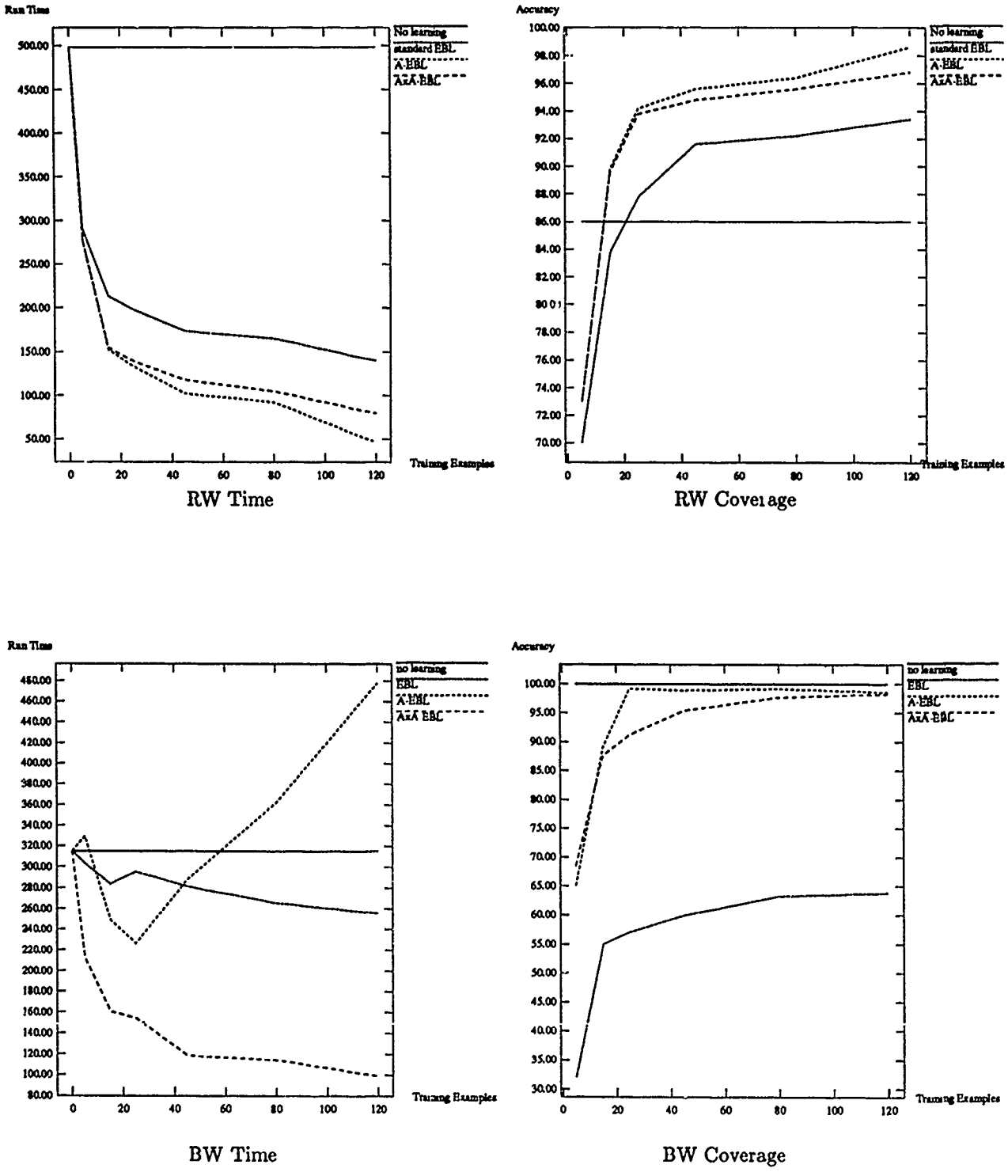


Figure 2: Performance and Coverage Curves for RW and BW

Table 2: Summary of experimental results

Domain	Average Time on Test Problems				AxA-EBL
	Nolearn	EBL	A-EBL	AxA-EBL	EBL
μLEX	2.019	0.103	0.103	0.104	1.01
RW	9.956	2.768	0.922	1.596	0.58
BW	3.150	2.553	4.774	0.987	0.39
GRID	0.137	0.361	0.423	0.268	0.74
Average	4.320	1.447	1.555	0.739	0.51

identically on the μLEX domain. This domain is not very informative for the purpose of comparing the techniques: it is included in this paper as an additional demonstration of the generality of AxA-EBL.

In the GRID domain, AxA-EBL starts out with a very bad set of control rules — one which usually leads the problem solver on a wild goose chase across the graph — but then quickly recovers. In all ten trials, AxA-EBL converges to the same program; the programs learned by EBL and A-EBL are still getting progressively slower. Unfortunately, the program learned by AxA-EBL is still slower than the original program. This indicates that some additional utility analysis may be necessary, even when approximations are used.

In the RW and BW domains, AxA-EBL outperforms EBL by wide margins. Suprisingly, A-EBL does somewhat better than AxA-EBL in the RW domain. We interpret this result as follows. The coverage graphs show that although AxA-EBL generates control rules which have lower match cost than A-EBL,[5] AxA-EBL converges somewhat more slowly; this is probably the price of searching through a much larger space of possible control rules. In short, *AxA-EBL trades off convergence speed for lower match cost* in the planning domains. This unfortunately leads to a performance tradeoff: if matching rules is cheap and problem solving is expensive, as in the RW domain, then faster convergence will offset the poorer quality of the rules, and A-EBL will produce faster programs.

5 Related work

Use of approximations in learning control knowledge was investigated in the MetaLEX project [Keller, 1987]. The techniques used in MetaLEX, however, were specific to state-space search, and required periodic testing of programs including learned control knowledge against a benchmark. This would be prohibitively expensive in a real-world domain. Ellman [Ellman, 1988], Tadepalli [Tadepalli, 1989] and Chien [Chien, 1989] have also investigated explanation based learning of approximate theories. The search techniques described by these authors improve on Keller's by taking advantage of a partial order on approximate theories; however, they have not been applied to the problem of learning control rules.

Approximations to EBL rules have also been investigated in [Chase *et al.*, 1989] in the context of the ULS system. The approximations done in ULS are not, however, selected by an inductive learning mechanism; as a consequence, ULS is limited to conservative approximations (*e.g.*, dropping one or two conditions). ULS nonetheless has made modest improvements in planning time in the RW domain.

The use of k-bounded approximations is one way of bounding the cost of learned rules. An alternative approach is described by Tambe and Rosenbloom in [Tambe and Rosenbloom, 1989]; they describe a syntactic constraint on SOAR programs which ensures that chunks have match cost linear in their length. A corresponding constraint can be imposed on EBL rules; it is easy to see that requiring every operational predicate to have at most one solution will lead to EBL rules which can be matched in linear time. An advantage to linear-time restricted EBL is that special-purpose inductive techniques are not needed to search for correct rules; the standard incremental learning algorithm can be used.

In order to satisfy the constraint that every operational predicate has at most one solution, it is typically necessary to change the operationality predicate. In the planning domains, for instance, it is sufficient to modify the operationality predicate so that the *holds* predicate, which determines if a condition is true in a state, is marked as non-operational. Using the new operationality predicate, control rules learned by EBL are expressed in terms of constraints on elements of the list structure used to represent a state, rather than constraints on the conditions that are true or false in that state. Changing the operationality predicate in this way closely corresponds to Tambe and Rosenbloom's suggestion of replacing multi-attributes in SOAR programs with list-processing utilities.

Figure 3 shows the result of using linear-time restricted EBL (LR-EBL) in the BW and RW domain. This experiment points out a disadvantage of the tech-

[5] This is obvious in the blocks world domain, since AxA-EBL's control rules outperform A-EBL's control rules even though their coverage is usually worse. It is a little more difficult to verify in the RW domain. One way to control for the effect of coverage is to compare only sets of control rules with equal coverage; this comparison shows that AxA-EBL control rules are about 12% faster than A-EBL control rules with equivalent coverage. Another indication that the AxA-EBL control rules are faster is that the best control rules learned by AxA-EBL outperform the best control rules learned by A-EBL, requiring 1.5 seconds versus 12.9 seconds to solve the 50 problems in the test set.

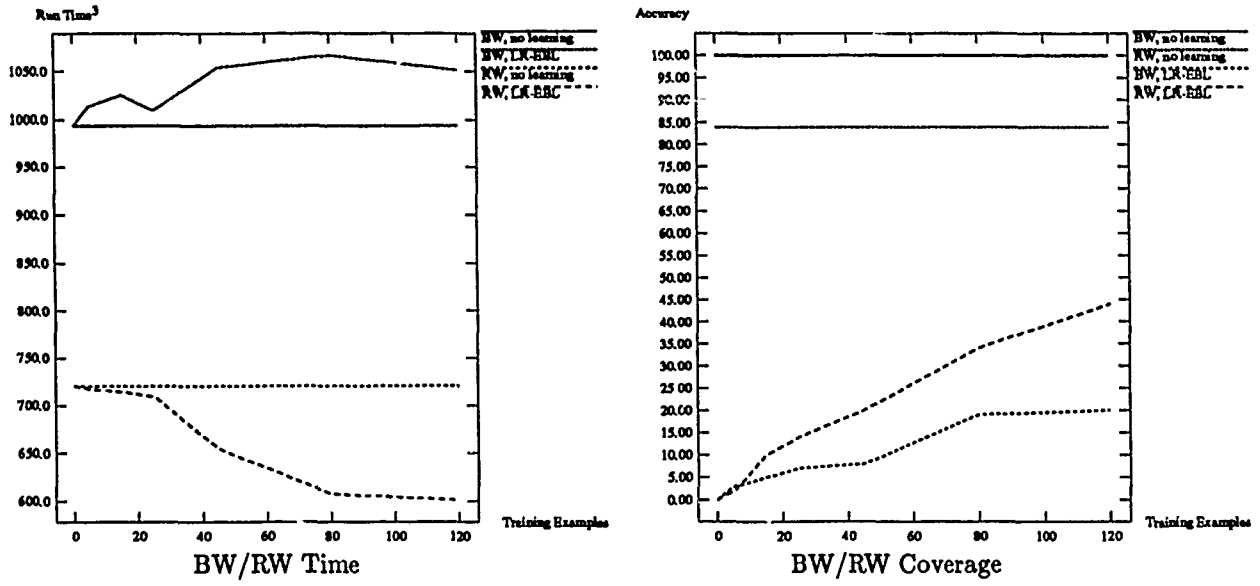


Figure 3: Performance and Coverage Curves for BW and RW with Linear-Time Rules

nique: learned rules tend to be very specific, and hence do not often transfer to new situations. As a consequence, there is little or no performance improvement on a test set of novel problems; and so in both domains, performance improvement was far less than that attained by AxA-EBL. In the RW domain, a modest 15-20% improvement gain was achieved, whereas AxA-EBL improved performance by over 80%; in the BW domain, performance was degraded slightly, whereas AxA-EBL improved performance by 65%.[6]

6 Conclusions

The utility of a rule is its contribution to performance improvement; utility is directly proportional to the coverage of a rule and inversely proportional to the match cost. This paper introduces two new techniques for improving the utility of learned rules. The first technique is to combine EBL with *inductive learning techniques*, thus increasing the coverage of control rules; the second technique is to constrain these inductive techniques to learn *approximate control rules* that have very low match cost. These two techniques have been synthesized in the AxA-EBL algorithm. AxA-EBL has been experimentally shown to produce control rules which improve substantially over the control rules learned by standard EBL in several domains; the average improvement is about a factor of two over the domains that have been studied.

Several open problems remain. First, the techniques used are non-incremental; it would be desirable if learning could take place concurrently with problem-solving, rather than as a separate pass. Second, the current implementation of AxA-EBL must, for reasons of computational complexity, use a very small and constraining set of approximations; it would be desirable to use a larger set of approximations. Third, one cost of learning approximate control rules seems to be a slower convergence rate, relative to simpler learning systems. It would be desirable if this tradeoff could be lessened or avoided.

[6]The training and test problems used for these experiments and the experiments of Figure 2 are the same; the CPU time needed to solve the test set without learning differes because with a different operationality predicate, different domain-specific predicates can be compiled.

Acknowledgements

I am grateful to my advisor, Alex Borgida, for his advice and encouragement; to Haym Hirsh, for invaluable suggestions on experimental methodology; to Chun Liew, for comments on a draft of the paper and several useful insights; and to Susan Cohen for proofreading a late draft of this paper. The author is supported by an AT&T Fellowship.

References

[Chase *et al.*, 1989] Melissa Chase, Monte Zweban, Richard Piazza, John Burger, Paul Maglio, and Haym Hirsh. Approximating learned search control knowledge. In *Proceedings of the Sixth Inter-*

national Workshop on Machine Learning. Morgan Kaufmann, 1989.

[Chien, 1989] Steve Chien. Using and refining simplifications: Explanation-based learning of plans in intractible domains. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* IJCAI, 1989.

[Cohen, 1989] William W. Cohen. Abductive explanation based learning: A solution to the multiple explanation problem. Technical Report ML-TR-26, Rutgers University, 1989.

[Cohen, 1990] William W. Cohen. Learning from textbook knowledge. In *Proceedings of the Eighth National Conference on Artificial Intelligence.* AAAI, 1990.

[Ellman, 1988] Tom Ellman. Approximate theory formation: An explanation-based approach. In *Proceedings of the Seventh National Conference on Artificial Intelligence.* Morgan Kaufmann, 1988.

[Fikes *et al.*, 1972] Richard Fikes, Peter Hart, and Nils Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[Flann and Dietterich, 1989] Nicholas Flann and Thomas Dietterich. A study of explanation-based methods for inductive learning. *Machine Learning*, 4(2), 1989.

[Keller, 1987] Richard Keller. The role of explicit contextual knowledge in learning concepts to improve performance. Technical Report ML-TR-7, Rutgers University, 1987.

[Markovitch and Scott, 1989] Shaul Markovitch and Paul Scott. Utilization filtering: A method for reducing the inherit harmfulness of deductively learned knowledge. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* IJCAI, 1989.

[Minton, 1988] Steven Minton. Learning effective search control knowledge: An explanation-based approach. Technical report, Carnegie-Mellon University Department of Computer Science, 1988.

[Mitchell, 1982] Tom Mitchell. Toward combining empirical and analytical methods for inferring heuristics. In *Human and Artificial Intelligence.* Lawrence Erlbaum, 1982.

[Mooney, 1989] R. J. Mooney. The effect of rule use on the utility of explanation based learning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* Morgan Kaufmann, 1989.

[Nilsson, 1987] Nils Nilsson. *Principles of Artificial Intelligence.* Morgan Kaufmann, 1987.

[Prieditis and Mostow, 1987] Armand Prieditis and Jack Mostow. PROLEARN: Towards a prolog interpreter that learns. In *Proceedings of the Sixth National Conference on Artificial Intelligence.* AAAI, 1987.

[Shavlik, 1987] Jude Shavlik. Generalizing number in explanation based learning. Technical Report UILO-ENG-87-2276, Univ. of Illinois/Champaign, 1987.

[Tadepalli, 1989] Prasad Tadepalli. Lazy explanation-based learning: A solution to the intractible theory problem. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* IJCAI, 1989.

[Tambe and Rosenbloom, 1989] Milinde Tambe and Paul Rosenbloom. Eliminating expensive chunks by restricting expressiveness. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* Morgan Kaufmann, 1989.

Applying Abstraction and Simplification to Learn in Intractable Domains

Nicholas S. Flann*
Department of Computer Science,
Oregon State University,
Corvallis, Oregon 97331-3902.
flann@cs.orst.edu,
(503)-737-4471

Abstract

In many domains it is computationally intractable to apply EBL to learn knowledge that is operational, complete and correct. Current approaches to solving this problem exploit the trade-off between correctness and operationality, and introduce approximations to learn operational but incorrect knowledge. In this paper we introduce an alternative approach that exploits the trade-off between completeness and operationality to learn knowledge that is both operational and correct, but *incomplete*, since it only covers a subset of the original domain. We employ a "boot-strapping" approach to incrementally increase coverage of the domain by using the knowledge learned from small problems to simplify learning from more complex problems. Correct knowledge is learned by employing a second order theory of goal achievement to construct abstract proofs that are useful for learning. These proofs are compiled into operational form by employing simplifying assumptions. The advantage of this approach is that learned knowledge is guaranteed to be correct if the simplifying assumptions made during learning hold during problem solving. We illustrate the method in two-person games and present preliminary results in Quinlan's lost-in-nply classification problem (1983).

1 Introduction

In optimization and 2 person game domains it is computationally intractable to apply EBL to learn knowledge that is operational, correct and complete. This is an instance of the *intractable theory problem* (Mitchell, Keller,& Kedar-Cabelli 1986; Tadepalli, 1989). The problem arises in these domains because of the resulting complexity of two steps in the EBL method: (a) constructing a complete proof and (b) extracting a correct and operational sufficient condition from the proof.

- Generating the proof is intractable. In optimization problems, to prove that the best possible goal has been achieved requires demonstrating that all other options lead to a result of lower value. In games, in addition to optimizing the goal achieved, we must also demonstrate that the goal will be achieved under all possible defensive actions by the opponent. Hence, in the worst case, proving that a goal is achieved requires exploring an exponential number of actions (in the depth of the proof).
- Analyzing the explanation is intractable. The goal of this analysis step is to extract a sufficient condition from the proof that is both operational and correct. This is difficult in the domains of interest because there is a strong tradeoff between correctness and operationality. By operational, we mean that the sufficient condition must be directly evaluable in the current situation. Hence the operator applications in the proof must be excluded from the sufficient condition. When the proof only includes existentially quantified operators this is straight forward (Hirsh, 1987). However, the proofs we are considering include *universal quantification* over operators. There is no simple solution to this problem (such as treating the $\forall$ as an "and" node) since the quantification is over *all possible operators* not just those that occurred in the particular example.

The principal approach to solving this problem has been to exploit the trade-off between correctness and efficiency and introduce approximations to learn incorrect but efficient knowledge (Ellman, 1988; Chien, 1989; Tadepalli, 1989). In Tadepalli

*I am grateful to my advisor, Tom Dietterich, for his advice and encouragement, and to Prasad Tadepalli for many interesting discussions and useful comments on a draft of this paper.

(1989), approximations are introduced by analyzing only incomplete proofs, while in Ellman (1988), approximations are introduced by simplifying the initial domain theory. These systems have demonstrated the usefulness of the approach in many domains including planning, chess end games, scheduling and hearts. However, one problem with these approaches is that the knowledge learned is incorrect and will produce errors when applied in the performance task. Moreover, because of the way the errors are introduced, the system does not know the conditions under which the compiled knowledge is correct and is therefore unable to determine if an answer provided is correct.

In this paper we present an alternative approach to learning in intractable domains. Rather then introduce approximations, we apply abstractions and simplifications to learn correct but *incomplete* knowledge. The knowledge learned is incomplete because it only applies to a subset of the complete domain. However, the knowledge is guaranteed to be correct for that sub-domain if the simplifying assumptions made during learning hold for that subdomain. Examples of simplifying assumptions include assuming that only a limited number of domain objects are in the problem situation.

The idea is to progressively cover the domain by learning correct knowledge for the simple cases first, then employing this knowledge to simplify the learning task for more complicated cases. This "bootstrapping" approach relies on a *transfer* of learned knowledge from simple cases to complex cases. The performance task will continue to make errors whenever the computational resources available are insufficient to solve a given problem. However, as more of the problem instances are solved using the more efficient learned knowledge, the overall number of errors made by the system should decline.

We illustrate our approach applied to two person games. In particular, we present preliminary results applying the approach to a classification problem in a sub-domain of chess: Quinlan's lost-in-n-ply problem (Quinlan, 1983).

The remainder of this paper is structured as follows: First, we detail our approach in a domain independent way. Second, we define Quinlan's lost-in-n-ply problem. Third, we describe the method in detail applied to learning in lost-in-2-ply. Fourth, we present preliminary empirical results for this subdomain. Fifth, we discuss related work. Finally, we conclude with a discussion of the current limitations of the approach and future work aimed at overcoming those limitations.

2 Approach

This discussion of our approach emphasizes the second of the two problems with intractable domains: extracting correct and operational sufficient conditions from proofs. The first problem—constructing the proof—is discussed later.

Before we describe our approach to solving this problem, it is instructive to consider why it is difficult to extract an operational and correct sufficient condition from the proofs constructed by min-max (or α-β) search. Consider, for example, the following min-max proof that the maximizing player, denoted +, has achieved an advantageous goal G in S in 2 ply:

(0) $achieve(G,S,\text{-},2) \Leftarrow$
$\forall\ Op_{-}, \exists\ Op_{+}, G(do(Op_{+},do(Op_{-},\text{S})))$

This rule can be read as "The maximizing player can achieve goal G in S in two ply because for all the minimizing player's moves (denoted Op_{-}) there exists a move for the maximizing player (denoted Op_{+}) that results in G being true." Note that we use a situation calculus encoding for operators where S denotes the initial situation and $do(Op,\text{S})$ denotes the situation following the application of the operator Op in S.

In order for the sufficient condition of this proof to be operational it must only test properties of the initial situation S. Therefore, since this proof currently tests properties of a future situation (in $G(do(Op_{+},do(Op_{-},\text{S})))$), we must replace this form with some equivalent (or sufficient) condition that only tests properties of S. If both the operators were existentially quantified, this could be achieved within the situation calculus framework by simply unfolding $G(do(Op_{+},do(Op_{-},\text{S})))$ until it terminated in literals that were true in S. The sufficient condition would then be the leaves of this proof (Hirsh 1987). However, this technique only applies when we have existential quantification, and if we apply it in (0) we will produce a sufficient condition that is over-general and therefore incorrect.

We present a new approach based on constructing alternative proofs that are more useful for learning. We construct these proofs by employing an abstract second-order theory that defines how goals can be achieved in terms of *influence relations* among operators, goals and situations. There are 3 primitive influence relations:

(1) $\eta(\lambda s.G(s),Op,S) \Leftrightarrow \neg G(S) \wedge G(do(Op,S))$
(2) $\delta(\lambda s.G(s),Op,S) \Leftrightarrow G(S) \wedge \neg G(do(Op,S))$
(3) $\rho(\lambda s.G(s),Op,S) \Leftrightarrow G(S) \wedge G(do(Op,S))$

where $\eta(G,Op,S)$ can be read as "if Op is applied in S then goal G will be made true," $\delta(G,Op,S)$ can be read as "if Op is applied in S then goal G will be made false," while $\rho(G,Op,S)$ can be read as "if Op is applied in S then goal G will be maintained true." Note the use of *lambda binding* for situations in the goals. This notation is employed because the relations are second order: they take a goal as an argument and evaluate it in two different situations, in the initial situation S and in the situation following the operator application $do(Op,S)$. We call

these primitives *influence relations* because they define the ways in which the application of operators affect the truth of goals. Note that the goals need not be simple literals in the domain theory like in the STRIPS encoding of domains, but may be arbitrarily complex expressions.

With these primitives, we can construct axioms that define how goals can be achieved through the application of operators. For example, the axiom below states the conditions under which a goal G is achieved when it is +'s turn to play in situation S:

$$(4)\ achieve(\lambda s.G(s), S, +, 1) \Leftrightarrow$$
$$G(S) \wedge \exists Op_+, \neg\delta(\lambda s.G(s), Op_+, S)$$
$$\vee \neg G(S) \wedge \exists Op_+, \eta(\lambda s.G(s), Op_+, S)$$

There are two cases: either G is already true and the operator does not affect it, or G is false and the operator makes the goal true. This rule can be used in counter planning situations to prove that G has been achieved when G is an advantageous goal for +. However, for complete counter planning we need to define the other case—when an advantageous goal is achieved following the opponent's (denoted -) turn to play:

$$(5)\ achieve(\lambda s.G(s), S, -, 1) \Leftrightarrow$$
$$G(S) \wedge \forall Op_-, \neg\delta(\lambda s.G(s), Op_-, S)$$
$$\vee \neg G(S) \wedge \forall Op_-, \eta(\lambda s.G(s), Op_-, S)$$

This axiom is basically the same as (4) but includes universal quantification over the operators. These axioms can in turn be used to produce proofs of goal achievement for any depth of ply by composing the two axioms. For example, the composition for achieving G in 2 ply when the opponent is first to move is:

$$achieve(\lambda s.G(s), S, -, 2) \Leftarrow$$
$$achieve(\lambda s_1.achieve(\lambda s_2.G(s_2), s_1, +, 1), S, -, 1)$$

We can produce proofs of goal achievement written in terms of the influence relations by unfolding the different cases of axioms (4) and (5). For example, the rule below is constructed by selecting the second case of (4) and combining it with the first case of (5):

$$achieve(\lambda s.G(s), S, -, 2) \Leftarrow$$
$$\exists Op_+, \eta(\lambda s.G(s), Op_+, S),$$
$$\wedge\neg\exists Op_-, \delta(\lambda s_1.\exists Op_+, \eta(\lambda s_2.G(s_2), Op_+, s_1), Op_-, S)$$

This rule can be read as: "+ achieves goal G in S in 2 ply because + is threatening to achieve the goal in 1 ply and the opponent cannot interfere." The first conjunct in the rule describes the threat of + achieving G, while the second conjunct describes -'s inability to prevent + from achieving G.

Other, more complicated rules can likewise be constructed for deeper ply, or for more than one goal. For example, the following rule proves that one of two maximizing goals is achieved in 2 ply:

$$achieve(\lambda s.[G_1(s) \vee G_2(s)], S, -, 2) \Leftarrow$$
$$\exists Op_+, \eta(\lambda s.G_1(s), Op_+, S),$$
$$\wedge\exists Op_+, \eta(\lambda s.G_2(s), Op_+, S),$$
$$\wedge\neg\exists Op_-, \delta(\lambda s_1.\exists Op_+, \eta(\lambda s_2.G_1(s_2), Op_+, s_1), Op_-, S),$$
$$\wedge\ \delta(\lambda s_1.\exists Op_+, \eta(\lambda s_2.G_2(s_2), Op_+, s_1), Op_-, S)$$

This rule is the familiar notion of a fork—when there is a double threat (first two conjuncts) and the opponent cannot simultaneously prevent both treats (last conjunct). In general, these rules provide a space of possible proofs for goal achievement in counter planning situations.

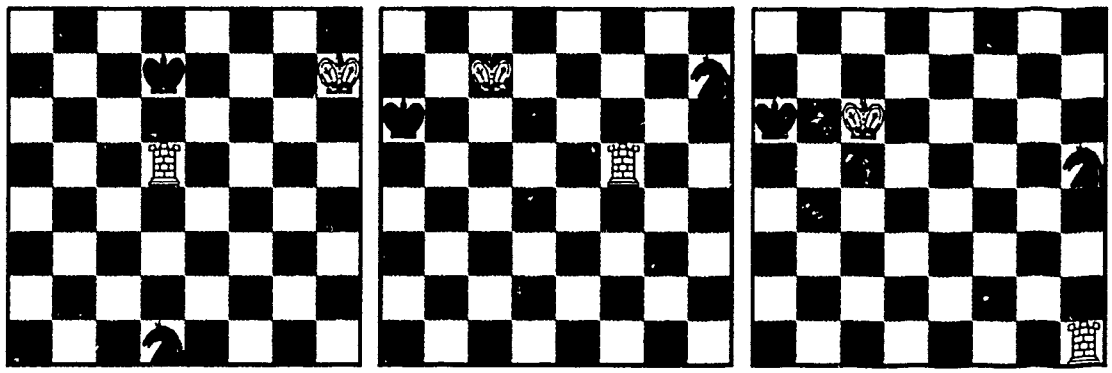
The important characteristic of these proofs which makes them suitable for learning is that they don't explicitly test properties of future situations. The only situation included in any of the literals in the proof is S, the initial situation. Hence, we can extract a sufficient condition from these proofs that only tests properties of the initial situation. However, we still do not have an "operational" sufficient condition because during match, we must evaluate nested and universally quantified influence relations—a computationally expensive task. Indeed, it appears that we have gained little by constructing the alternative proof. However, the alternative proofs differ from the min-max proofs in that the operators are not completely unconstrained—we need only consider *relevant* operators that *affect* the goals according to the influence relations included in the proof. In compiling the alternative proof for fork above we need only consider operators of - that *prevent* + from achieving each of the two goals.

We introduce a new approach to compiling these sufficient conditions that produces a simple pattern of features that can be efficiently tested in the initial situation. The approach exploits the *simplifying assumptions* mentioned earlier to reduce the complexity of the computation. These simplifications are constraints on the situation S such as restrictions on the number, geometrical arrangement, or properties of objects in S.

This concludes our description of the general approach. We now describe an application domain (Quinlan's lost-in-n-ply) and illustrate the method in detail.

3 Domain: Quinlan's lost-in-n-ply

Quinlan's lost-in-n-ply domain is a sub-domain of chess with only 4 pieces: knight and king against rook and king. The performance task is one of *classification*—given a position, determine if it is lost-in-n-ply for the knight side, where a loss is defined as the capture of the knight (without recapture of the rook) or check-mate. Although this domain is much simpler than full chess, it can present quite a challenge even to the master-rated player (Kopec & Niblett, 1980). This is a large domain that includes over 11 million legal knight-side-to-move positions and 9 million rook-side-to-move positions. In general, about half of the rook-side-to-move positions and one fifth of the knight-side-to-



state1 *state2* *state3*

Figure 1: Three problem instances from lost-in-2-ply, black to play, white to win

legal-move(S,do(Op,S),Side) ⇐
  pseudo-move(Op,S,Side),
  ∧ ¬*in-check(do(Op,S),Side)*

pseudo-move(op(FSq,TSq,[Type,Side],empty),S,Side) ⇐
  on(S,FSq,[Type,Side]),
  ∧ *legal-direction(Type,Dir)*,
  ∧ *reachable(S,FSq,TSq,Type,Dir)*,
  ∧ *on(S,TSq,empty)*

lost(S,Side) ⇐
  kn-l(S,Side)
  ∨ *check-mate(S,Side)*

check-mate(S,Side) ⇐
  in-check(S,Side),
  ∧ ∀ *Op*, *pseudo-move(Op,S,Side)* ⇒
    in-check(do(Op,S),Side)

Table 1: Selected axioms from the chess domain theory

move positions are losses for the knight side. However, many of these losing positions are lost within small search horizons (i.e., small values of n). Below we give a break down of the percentage of positions that are lost-in-n-ply for small values of n:

Search depth n	Total count of lost positions	Percentage of total lost positions
1	3.03×10^6	58.0 %
2	0.76×10^6	15.0 %
3	0.37×10^6	7.0 %

We apply our learning approach to this problem by learning the simple cases (small values of n) first and then learning the more complicated cases. The simplifying assumption we employ is that all situations contain only the four playing pieces.

In the remainder of the paper we illustrate the method solving the following problem:

Given:

- A simple declarative encoding of chess legal moves and goals.
- Randomly drawn positive examples of lost-in-n-ply for some n.

Find:

- A small decision tree that efficiently and correctly classifies lost-in-n-ply positions.

We illustrate selected lost-in-2-ply positions in Figure 1 and illustrate selected axioms from the chess domain theory in Table 1.

4 Method

The learning method is applied whenever the current decision tree fails to classify a give problem instance. The method has three stages: (1) produce a proof of goal achievement, (2) produce an operational sufficient condition from the proof and (3) integrate this sufficient condition into the decision tree. We illustrate the method learning from problem instance *state1* in Figure 1.

4.1 Produce a proof of goal achievement

The goal of this stage is to produce a proof of lost-in-2-ply using the abstract theory and the domain theory for chess. Currently, the implementation constructs this proof by first constructing a min-max search tree then *reconstructing* a proof of lost-in-2-ply using axioms from the abstract theory. This reconstruction process is constrained by the min-max search tree in a manner similar to Minton's EBS process (Minton, 1988). In the final section of the paper we discuss an alternative approach that avoids first constructing the min-max search tree. The proof produced is given below:

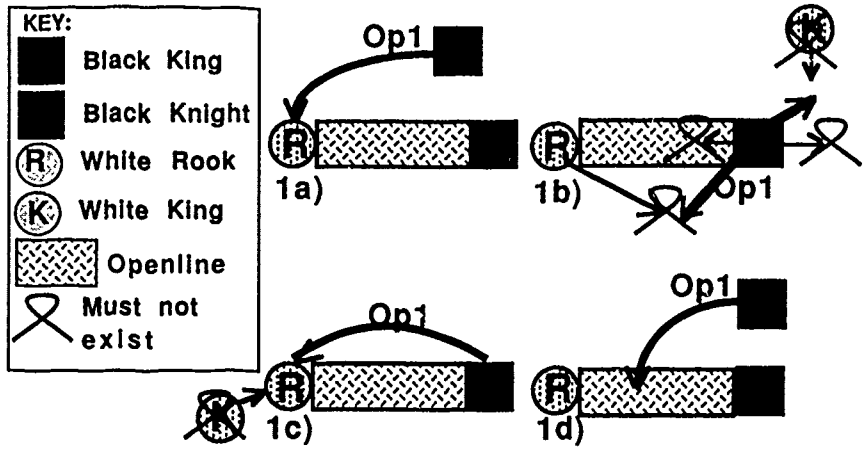


Figure 2: Compiling $\delta(\lambda s.in\text{-}check(s), Op1, S)$

$achieve(\lambda s.lost\text{-}in\text{-}2\text{-}ply(s), S, -, 2) \Leftarrow$
$in\text{-}check(S),$
$\wedge \exists Op_+, \eta(\lambda s.kn\text{-}l(s), Op_+, S),$
$\wedge \neg \exists Op_- \delta(\lambda s_1.\exists Op_+, \eta(\lambda s_2.kn\text{-}l(s_2), Op_+, s_1), Op_-, S),$
$\quad \wedge \delta(\lambda s.in\text{-}check(s), Op_-, S)$

This proof can be read as: "Black is lost in 2 ply because the black king is in check, white can capture black's knight and both black's goals of preventing the check threat *and* preventing the knight from being captured cannot be achieved." The *in-check* constraints arise from applying the following additional axiom from the abstract theory to the $\neg in\text{-}check(do(Op, S), black)$ constraint in the definition of *legal-move* in Table 1. The axiom states the conditions under which a goal G is guaranteed to be false following an operator (Op) application:

$\neg G(do(Op, S)) \Leftrightarrow$
$\quad G(S) \wedge \exists Op, \delta(\lambda s.G(s), Op, S)$
$\vee \neg G(S) \wedge \exists Op, \neg\eta(\lambda s.G(s), Op, S)$

4.2 Produce an operational sufficient condition

The goal of this stage is to find a sufficient condition of this proof that is efficient to evaluate. The target form is a small disjunction of conjunctions of *features*, where a feature is defined as a conjunction of "operational" (i.e., directly evaluable) literals.

The first two forms in the proof are easy to compile. Each one is used to define a new feature by extracting its sufficient condition from the example situation. The $\exists Op_+, \eta(\lambda s.kn\text{-}l(s), Op_+, S)$ becomes the $white\text{-}rook\text{-}attacks\text{-}knight(S)$ feature defined[1]:

$white\text{-}rook\text{-}attacks\text{-}knight(S) \Leftrightarrow$
$\quad on(S, SqR, [rook, white]),$
$\quad \wedge\ on(S, SqN, [knight, black]),$
$\quad \wedge\ openline(S, SqR, SqN, Dir),$
$\quad \wedge\ legal\text{-}direction(rook, Dir)$

This feature is true in a situation S when the white rook is on SqR, the black knight is on square SqN and there exists line of empty squares between SqR and SqN along direction Dir such that Dir is a legal direction for the rook.

This results in the following sufficient condition:

$achieve(\lambda s.lost\text{-}in\text{-}2\text{-}ply(s), S, -, 2) \Leftarrow$
$white\text{-}rook\text{-}attacks\text{-}knight(S),$
$\wedge white\text{-}rook\text{-}checks\text{-}king(S),$
$\wedge \neg \exists Op_-, \delta(\lambda s_1.\exists Op_+, \eta(\lambda s_2.kn\text{-}l(s_2), Op_+, s_1), Op_-, S),$
$\quad \wedge \delta(\lambda s.in\text{-}check(s), Op_-, S)$

The more challenging task for the compiler is to compile the influence constraint over Op_- into features. The system first re-expresses the form as universals:

$\forall Op_1, \forall Op_2, \delta(\lambda s_1.\exists\ Op_+ \eta(\lambda s_2.kn\text{-}l(s_2)\, Op_+, s_1), Op_1, S),$
$\quad \wedge\ \delta(\lambda s.in\text{-}check(s), Op_2, S) \Rightarrow$
$\quad Op_1 \neq Op_2$

The universals reveal the problem with compiling this form—we must consider all possible Op_1's and Op_2's in the influence relations. To simplify this process, we exploit all the constraints that apply to S including contextual constraints and simplifying assumptions. First, we know that both $white\text{-}rook\text{-}attacks\text{-}knight(S)$ and $white\text{-}rook\text{-}checks\text{-}king(S)$ are true in S. Second, we apply the simplifying assumption and assume that the 4 pieces (black knight and king, white rook and king) are all the pieces that *can ever occur* in S.

Given these constraints we first compile the individual influence relations. In order to understand how this can be achieved, it is useful to review the definition of the influence primitives defined previously. Given a situation S in which G is true, $\delta(\lambda s.G(s), Op, S)$ must generate operators that when

[1]The system simply generates "gensym" names for the features, I have used intuitive names to assist understanding.

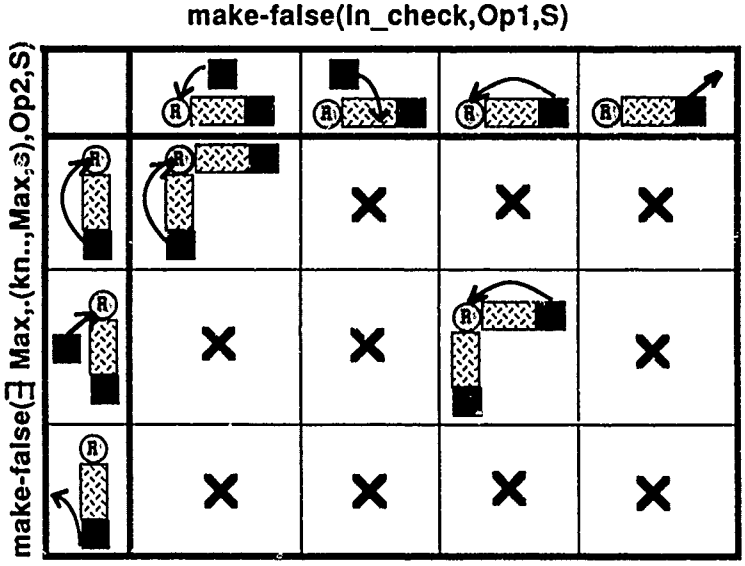


Figure 3: Compiling $\forall Op_1, \forall Op_2, \delta(\lambda s.G_1(s), Op_1, S) \wedge \delta(\lambda s.G_2(s), Op_2, S) \Rightarrow Op_1 \neq Op_2$

applied in S make G false. If we were working in a domain where the goals are simple literals directly modified by the operators (i.e., the goals are literals in the add and delete lists), computing *make-false* would be easy; we simply identify those operators that have G both in the delete list and precondition. However, this process is more complicated when G is some *derived property* of the situation.

We employ an eager partial evaluation technique that generates all possible operators in S *symbolically* and determines those that result in G being false. The result of this analysis for $\delta(\lambda s.in\text{-}check(s), Op1, S)$ is illustrated graphically in Figure 2. Note that the technique produces 4 sets of operators: (1a) where the black knight takes the rook, (1b) where the black king moves out of check, (1c) where the king takes the rook, and (1d) where the knight blocks the check. Note that additional constraints are introduced to ensure that the goal will be false following the operator application. When moving the king out of check, the direction must not be along the line of the check, nor must either the rook or the white king be in a position to check the black king in its destination square.

Applying the technique to the $\delta(\lambda s_1.\exists\ Op_+ \eta(\lambda s_2.\ kn\text{-}l(s_2)\ , Op_+, s_1), Op2, S)$ similarly yields 3 sets of operators: (2a) where the black knight moves out of the way, (2b) where the black king takes the rook and (2c) where the knight takes the rook. Again, additional constraints are included so that the goal is false following the operator application.

The final stage of compiling this expression into features is to eliminate the $\forall Op_1$, $\forall Op_2$ expression. To achieve this we do exhaustive case analysis by "multiplying out" the two sets of operators. We unify each operator in set (1) with each operator in set (2) and retain those that are consistent. This process[2] is illustrated in Figure 3.

To generate the sufficient condition we first define features from the resulting patterns (in Figure 3), then negate the features (since the intersection of the operators must be empty). Simplifying the result produces the following operational sufficient condition:

achieve($\lambda s.$*lost-in-2-ply*$(s), S, -, 2) \Leftarrow$
[*white-rook-attacks-knight*(S),
$\wedge$ *white-rook-checks-king*(S),
$\wedge$ *black-king-attacks-rook*(S),
$\wedge$ *white-king-protects-rook*(S)]
$\vee$[*white-rook-attacks-knight*(S),
$\wedge$ *white-rook-checks-king*(S),
$\wedge \neg$ *black-king-attacks-rook*(S)]

4.3 Update the classification procedure

The final stage of learning is to update the current decision tree. This is achieved by employing an incremental version of ID3 such as ID5 (Utgoff, 1988). The final decision tree produced for lost-in-2-ply is illustrated in Figure 4 (the new subtree is circled on the left).

5 An evaluation in lost-in-n-ply

We consider three evaluation criteria, two that evaluate the performance component and one that evaluates the learning method. To evaluate the performance component we consider (1) the relationship

[2] Note that this process produces two cases, one where the knight takes the rook. In fact, this case is impossible due to geometry and is eliminated by a later stage.

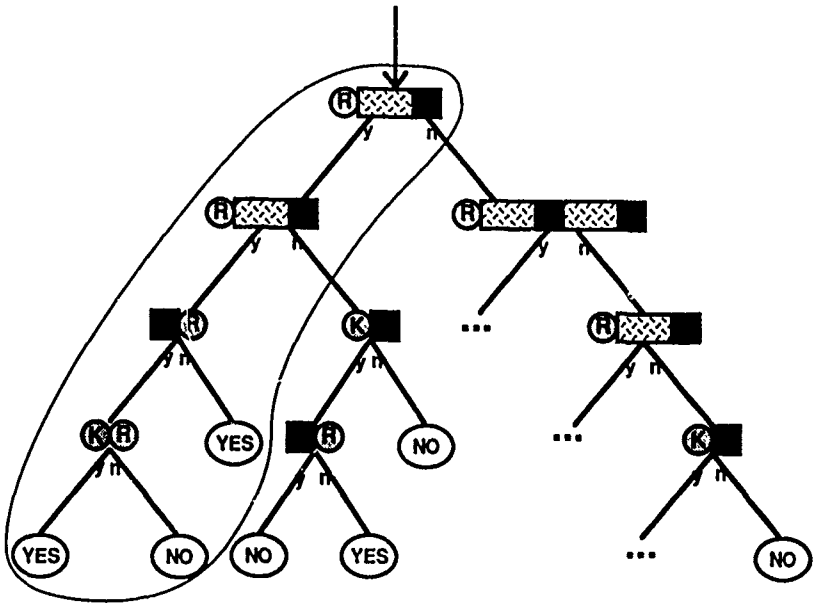


Figure 4: The updated decision tree

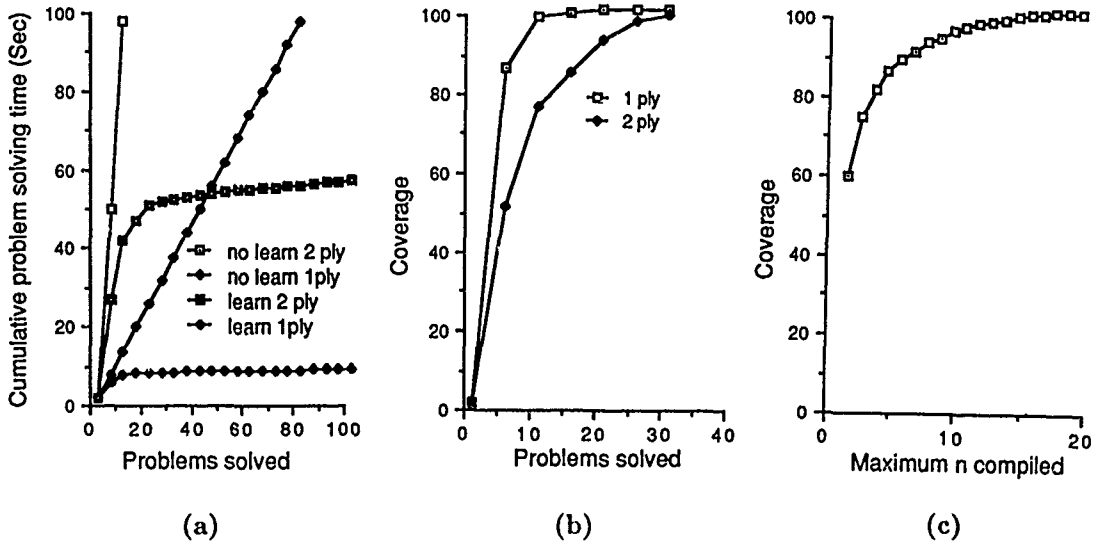


Figure 5: Evaluation in lost-in-n-ply. Graph (a) illustrates the cumulate problem solving time as a function of the number of examples processed, for fixed n, (b) illustrates the coverage of the decision tree as a function of the number of examples processed, for fixed n, and (c) illustrates the overall coverage as a function of the maximum n (search depth) compiled.

between the problem solving time and the number of training examples processed, and (2) the relationship between the domain coverage of the decision tree learned and the number of tra ining examples processed. To evaluate the learning method we consider (3) the relationship between the time spent learning and the proportion of the domain covered by the decision tree.

1. **Recognition time / Instances processed:** We performed an empirical study for lost-in-n-ply: Let I_n be the set of all positive instances of lost in exactly n ply. For each n we incrementally learn a decision tree from randomly chosen examples from I_n. Let T_n be a set of 100 randomly drawn instances from I_n. After each learning event (i.e., an update of the decision tree) we used the tree to classify those examples in T_n. During experimentation we recorded both the cumulative problem solving time and the average time to solve the problems in T_n. In Figure 5(a) we report the cumulative problem solving time for the decision tree[3] for each fixed n. The asymptotic results for T_n are given below:

Search depth	Average Problem solving time	
n	No Learn	Decision Tree
1	$1.2S$	$5.0mS$
2	$10.0S$	$30.0mS$

The Decision Tree column for $n = 2$ gives the results for the decision tree illustrated in Figure 4.

2. **Coverage / Instances processed:** We repeated the above experiment, this time recording the percentage of T_i that are classified by the decision tree. These results are included in Figure 5(b). In Figure 5(c) we give the overall coverage for the complete domain under the condition that for all $i, i \leq n$, lost-in-i-ply has achieved 100 % coverage.

3. **Learning time / Coverage:** In the current implementation, to learn from a lost-in-n-ply example, the system must construct a min-max search tree of depth n. Since the complexity of constructing this proof is exponential in n, the learning time grows exponentially as more of the domain is covered and complete coverage is impossible. We will return to this issue in the final discussion section.

6 Related Work

Minton (1984) introduced an explanation-based technique for learning plans in games. The method learns rules that recognize opportunities for achieving goals via sequences of forced moves. This approach was applied successfully to Go-moku and Tic-Tac-Toe. However, it was not satisfactorily applied to chess or other complex games. One reason for this is that the system employed a very restricted representation language for describing the forcing conditions underwhich a goal is achieved. It was difficult to represent conditions such as "for all empty squares *sq*, between the my king and the attacking piece, there does not exist a piece that can move into *sq*." The method was further limited since it was designed to learn from only one particular kind of forced loss—the simple fork.

Tadepalli (1989) introduces an approach to learning in chess end games based on learning optimistic plans that are incorrect. These are then incrementally refined upon failure. This approach and the one described here can be viewed as opposite sides of the lazy/eager trade-off. My approach eagerly computes all *relevant* interactions at compile time, while Tadepalli's approach is lazy, it assumes there are no interactions between plans. The principal disadvantage with the lazy approach is that a burden is placed on the human trainer to correct the errors introduced by the approximations. My approach prefers to expend cpu time over human trainer time. However, there is a danger that learning will become intractable. Ultimately, some mixed strategy may be appropriate.

Quinlan (1983) applied the inductive learning algorithm ID3 to learn a decision tree for small values of n in lost-in-n-ply. This work demonstrated that successful learning relies critically on the choice of instance vocabulary. Quinlan spent a considerable amount of time "hand engineering" sets of features for lost-in-2-ply (2 man weeks) and lost-in-3-ply (over 3 man months), and he gave up on lost-in-4-ply. In contrast, the approach here needs no special engineering; the chess domain theory provided is very simple and succinct. Hence, this approach reduces the need to perform "vocabulary engineering" to achieve successful learning.

Braudaway and Tong (1989) describe a compilation method that is similar to the one described here. The most significant similarity is the use of *abstract case analysis* to perform the compilation. In the work described here, the expressions that describe sets of operators (such as those operators that move the king out of check) can be thought of as abstract cases. Compilation consists of enumerating all possible abstract cases within a sufficient condition and determining the interactions among them. This process is well illustrated in Figure 3. Braudaway and Tong use a similar process to compile a declarative specification of a legal floor plan into an efficient generator of legal designs. Here, the abstract cases are partial solution generators and compilation involves simulating the generators to determine their

[3] Due to the recency of this work, only the $n = 1$ and $n = 2$ cases have been completed.

interactions. One principle difference between the two works is that in Braudaway and Tong's system, the abstract cases arise from an explicit hierarchy of structured objects (such as lines, corners, rectangles etc), while in the work described here, the abstract cases arise from partially evaluating the influence relations.

7 Discussion

The evaluation strongly suggests that the method can be effective. However, there are two principal drawbacks with the current implementation: (a) the performance task is limited to classification—a more useful performance task would be to apply the learned knowledge to actually solve problems (i.e., play games), and (b) proof construction is still intractable. In this section we briefly discuss a solution to both these problems that is currently under investigation.

Both these difficulties stem from the same problem: using the abstract theory only to *explain* problem solving and not to *construct* problem solving. This approach is understandable initially, since the abstract theory is extremely under-constrained when used in a wholly backward or top-down manner. However, once learning has complied some of the proofs to patterns, a more forward or bottom-up approach may be appropriate. For example, *state3* in Figure 1 could be solved effectively bottom-up using the fork rule (given earlier in the Approach section) once *white-rook-attacks-knight* has been learned from *state1* and *white-rook-threatens-checkmate* has been learned from *state2*.

By constructing the proof bottom-up we can play games by selecting at each turn the operator that leads to the best goal. The bottom-up approach can also overcome the problem with intractable proof construction, since constructing a proof for lost in $n+1$ ply can exploit *transfer* from the already compiled proofs of lost in i ply, for $1 \leq i \leq n$.

This paper has reported preliminary results for this new approach. In addition to exploring the use of the theory for constructing proofs and completing the lost-in-n-ply problem, other work in progress includes: (1) applying the technique to other subdomains of chess including some of the standard endgame databases (Bratko & Michie, 1980), (2) applying the technique to other game domains including checkers, variants of chess and Go-moku, (3) applying the technique to some non-game domains such as such as scheduling, and (4) developing a theory of the method that formalizes the expected behavior in terms of characteristics of the application domain.

8 Bibliography

Bratko, I. and Michie, D. (1980). A Representation for pattern-knowledge in chess endgames. In M. R. Clarke (Ed), *Advances in Computer Chess.*

Braudaway, W. and Tong, C. (1989). Automated synthesis of constrained generators. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence,* Detroit, Michigan.

Chien, S. (1989). Using and refining simplifications: Explanation-based learning of plans in intractable domains. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.*

Ellman, T. (1988). Approximate theory formation: An explanation-based approach. In *Proceedings of the Seventh International Conference on Artificial Intelligence,* St. Paul, MI.

Hirsh, H. (1987). Explanation-based generalization in a logic programming environment. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence,* Milan.

Kopec, D. and Niblett, T. (1980). How hard is it to play the king-rook king-knight ending? In M. R. Clarke (Ed), *Advances in Compt ter Chess.*

Minton, S. (1988). *Learning effective search control knowledge: An explanation-based approach.* Ph.D. Th., Computer Science Department, Carnegie Mellon University, March 1988.

Minton, S. (1984). Constraint-based generalization: Learning game-playing plans from single examples. In *Proceedings of the Third International Conference on Artificial Intelligence,* Austin, TX.

Mitchell, T., Keller, R. and Kedar-Cabelli, S. (1986). Explanation Based Generalization: A Unifying View. In *Machine Learning,* Vol. 1.

Quinlan, J., R. (1983). Learning efficient classification procedures and their application to chess end games. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* San Mateo, CA: Morgan Kaufmann.

Tadepalli, P. (1989). Lazy explanation-based learning: A solution to the intractable theory problem. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence,* Detroit, MI.

Utgoff, P., E. (1988). ID5: An incremental ID3. In *Proceedings of the Fifth International Conference on Machine Learning,* pp 107-120.

Explanation-Based Learning with Incomplete Theories: A Three-step Approach

Jean Genest†
Dept of Mathematics
Collège Militaire Royal de St-Jean
Richelain, Québec, J0J 1R0,Canada
(514) 358-7011

Stan Matwin
Dept of Computer Science
University of Ottawa, Ottawa,
Ontario, K1N 6N5, Canada
(613) 564-5069
stan@uotcsi2.bitnet

Boris Plante
Dept of Computer Science
University of Ottawa, Ottawa,
Ontario, K1N 6N5, Canada
(613) 564-5069

Abstract

A weakness of EBL is its inability to explain when the theory is incomplete. This paper presents a three-step approach to deal with incomplete theories based on abduction, analogical reasoning and case-based reasoning. Abduction allows us to explain an example in the context of an incomplete theory, without modifying the theory. The simple variant of analogical reasoning used here not only provides an explanation, but also extends the domain theory. We show that the overhead imposed by our method on EBL is acceptable (at most polynomial). The approach presented in this paper was implemented in a system called LISE (Learning in Software Engineering). LISE is a system which translates informal and non-operational user requirements into formal and operational software specifications.

Keywords: Explanation-Based Learning, Incomplete Theory, Abduction, Analytical Cost Evaluation

1 Introduction

This paper addresses the problem of learning in the EBL framework [Mitchell et al. 1986], [DeJong et al. 1986], [Ellman 1989] with an incomplete domain theory. The proposed strategy applies three procedures to an incomplete explanation in order to plausibly complete it. The procedures applied are: abduction, analogical reasoning and case-based reasoning. More specifically, when dealing with an incomplete explanation, we are first trying to apply abduction to the missing part of the explanation. If this fails, analogical inference is applied to complete the explanation, and should this fail to make the explanation complete - case-based reasoning is used.

The three-step approach was implemented in a system called LISE (Learning in Software Engineering). LISE is a system using EBL with an incomplete domain theory to translate informal and non-operational user requirements into formal and operational software specifications. Examples from LISE will be used in this paper to illustrate the three-step approach.

LISE was successfully applied to the design of a specification for a banking system and a fleet management system.

2 The Three-step Approach To Deal With Incomplete Theories

An explanation in EBL is built using rules. The antecedents of the rules are satisfied using facts from the training example or using the consequents of other rules. When the domain theory is incomplete, rules that would be required to complete a particular explanation might be missing. If it is the case, EBL will produce one or many partial explanations. A partial explanation is an explanation containing proven and unproven antecedents. The unproven antecedents are antecedents for which no facts were found in the training example, and which cannot be proven in the existing domain theory.

An incomplete domain theory is recognized when one or many partial explanations are produced instead of a complete explanation. Partial explanations are built using rules which have in their antecedents some predicates in common with the training example facts. These rules are used in our approach to build a plausible explanation of the training example.

There are three steps in our approach to deal with the incomplete domain theory. The first step starts by selecting the partial explanation providing the best coverage of the training example. Next, abduction is used to complete the partial explanation so that a plausible explanation be produced without having to extend the domain theory.

The second step in our approach is applied when the first step does not work. It starts by selecting the partial explanation providing the best coverage of the training example. Next, a plausible explanation is created using analogical reasoning applied between the unproven

† This work was done at the Knowledge Acquisition Laboratory, University of Ottawa. The three authors are with the Ottawa Machine Learning Group (OMLG).

antecedents of the partial explanations and the training example facts. Multiple partial explanations can also be combined in the creation of the plausible explanation. Finally, new rules are extracted from the plausible explanation and added to the domain theory.

The third step in dealing with an incomplete domain theory involves the usage of a case-based system. It will be applied only if the two previous steps did not work. In this step, the case-based system will retrieve a case and adapt it to the training example. The domain theory will not be extended in this step.

2.1 Selecting The Best Partial Explanation

The abductive and analogical reasoning steps require that the best partial explanation be selected. We define the best partial explanation to be the partial explanation which provides the best coverage of the training example according to a heuristic we developed.

The analogical reasoning step may also require that one or more additional partial explanations be selected when a combination of partial explanations is required to build the plausible explanation. We addressed that requirement by ranking partial explanations generated for a specific training example according to a score. The score is attributed to each partial explanation based on its coverage of the training example. The heuristic developed to calculate the score is as follows:

a. reward a partial explanation for each common feature it shares with the training example,
b. reward concise partial explanations, where conciseness is measured in terms of inferences required to prove a goal (the less inferences in the explanation, the more concise it is),
c. penalize a partial explanation for each of its unproven leafs,
d. penalize a partial explanation for each feature of the training example that was left unaccounted for, and,
e. penalize slightly a partial explanation for each abductive inference that was used in its construction.

Our heuristic is based mainly on the syntactic nature of the partial explanations. We are currently investigating the validity of the heuristic from a cognitive science viewpoint. We anticipate that a semantic measure will be appropriate, so that the ranking of partial explanations also takes into account the goals of the users. Such a semantic measure will require the use of background knowledge in the ranking of the partial explanations.

2.2 Abduction To Complete Partial Explanations

Abduction is the generation of hypotheses, which, if true, would explain observed facts [Pople 1973]. More precisely, if the rule `Q <- P` and the fact `Q` are given, then the desired abductive conclusion is `P`. `P` can be characterized as being an hypothesis because there could exist another rule `Q <- P'` which would have been used to derive `Q`.

Abduction will be used to complete the best partial explanation selected by our heuristic. Considering the training example features as facts, we will attempt to draw hypotheses from the unproven antecedents using abduction. If hypotheses can be drawn for each unproven antecedent, the partial explanation will be completed. Using the above example of the rule `Q <- P` and `Q, P` would be an unproven antecedent in the partial explanation and `Q` would be a fact given as a training example feature. If `P` is the only unproven antecedent, an hypothesis can be drawn to account for the occurrence of `Q` in the training example, and the partial explanation can be completed.

Abduction is not used to extend the domain theory. The unproven antecedents of the rules used to provide the partial explanations are changed into proven antecedent based on the abductive inference. There is no possibility that the correctness of the domain theory be jeopardized since no new rule is created.

2.3 Analogical Reasoning To Complete Partial Explanations

The analogical reasoning paradigm in our work consists of deductive inferences made using goals that are common to features. Two features having a common goal are said to be *analogous*. Goals are kept in the domain theory.

When using analogical reasoning, a plausible explanation is built by first re-using the proven antecedents of the best partial explanation as determined by the heuristic. Unproven antecedents of the partial explanation are replaced by analogous training example facts. New rules are extracted from the plausible explanation and will be added to the domain theory.

On certain occasions, more that one partial explanation might be required to build the plausible explanation. This is because a single partial explanation might explain some features of a training example while leaving out other features covered by another partial explanation. Our approach provides a mean of combining these multiple partial explanation into a single plausible explanation. Roughly, the proven antecedents of the partial explanations are re-used and the unproven antecedents are replaced by analogous training example facts. When partial explanations are combined, it sometimes happens that several unproven antecedents have no analogous features in the training example. An analysis of the goals of these unproven antecedents is performed to see if they can be substituted or removed.

2.4 Case-based Reasoning Applied To Training Examples

The domain theory contains rules which normally enable EBL to explain most positive training examples. However, some instances of concepts are not readily explainable using these rules. These instances are exceptions to general rules. These instances would normally be covered by a small disjunct according to [Holte et al. 1989]. They also correspond to the marginals of [Matwin et al. 1990]. The third step of our approach employs a case-based reasoning system to retrieve previous cases to apply to training examples which represent exceptions. A previous case is an extension to a concept definition provided by the incomplete theory.

The case-based system will retrieve a case for a training example if there is a match between the features of the case and the features of the training example, and if the order of the features of the case is preserved in the training example. There is a match between a case feature and a training example feature when thename of the case feature is the same as the name of a training example feature and the variables in the case unify with the constants in the training example. The unification is propagated to other features of the case. There will also be a match when the constants in case features are associated with the same constants in the training example. Finally, there is a match when a feature in the case can be associated with an analogous feature in the training example as long as all the previous conditions hold. When more than one case is retrieved, a heuristic similar to the one used to rank the partial explanations will select the most applicable one.

The cost of matching, the likelihood that a case be applicable, and the size of the case-base make the case-based approach secondary to the rule-based approach of EBL in our methodology to deal with the incomplete domain theory. An example of case-based reasoning to deal with the incomplete domain theory is in [Genest and Matwin 1990].

3 An Example: The System LISE

LISE (Learning In Software Engineering) is a system inspired by the learning process experienced by an analyst during the analysis phase. LISE uses EBL (Explanation-Based Learning) to explain positive training examples corresponding to user requirements using a domain theory corresponding to a specification. When a training example (an instance of user requirement) can be explained, the result is the corresponding specification expressed using primitive operations. When a training example can not be explained, LISE will apply our three-step approach to construct a plausible explanation using abduction, analogical reasoning or case-base reasoning.

3.1 The Domain Theory In LISE

In LISE, the domain theory represents the specification of an application. In the particular application considered here, the domain theory will be the specification of a banking system.

The specification of the system is given in terms of objects and operations applicable to objects. The objects represent structural properties of the system. The operations represent behavioural properties of the system. The format of our domain theory is inspired from the Extended Semantic Hierarchy Model (SHM+) described in [Brodie et al. 1984].

The domain theory consists of frames arranged in a hierarchy and allowing multiple inheritance of properties. Each frame specifies an object or an operation using a set of properties. One common property of all frames is `isa` which links a frame to its parents.

Frames representing objects are located under the high-level frame called `ENTITY`. Similarly, the frames representing operations are located under the frames `ACTION` and `TRANSACTION`. Each frame describing an operation is defined by a two special properties: *precondition* and *procedure*.

The precondition property specifies the condition under which the operation can be applied. The procedure property specifies a fixed sequence of operations. Operations specified as `ACTION` in the hierarchy are non-primitive operations. Non-primitive operations are defined by a precondition and a procedure which includes a single primitive operation. Primitive operations are atomic and they are not defined anywhere. Operations specified under `TRANSACTION` in the hierarchy are defined by a precondition and a procedure containing primitive and non-primitive operations arranged in a fixed sequence. Figure 1 illustrates six frames taken from a domain theory containing the specification of a banking system. Each frame of the domain theory is transformed into a rule prior to executing EBL.

The frames defining entities enumerate properties of the entity (e.g. person has a name). Some properties of entities and some actions possess goals. These goals are included in the domain theory of LISE. Goals are essential to the analogical reasoning process employed to extend the domain theory. An instance of a goal is:

```
goal(credit_margin(Person,Margin),
     protect_bank_interest)
```

which is read: the goal of the `credit_margin` property of a person is to protect the bank interest.

3.2 An Example Of Abduction

The next scenario illustrates the usage of abduction, introduced in sec. 2.2., to complete a partial explanation. The training example is the requirement for the transaction `withdraw(bob,100)` where the fact `account(bob,acc_1)` was replaced by `client(bob)` (Figure 2).

```
person
isa: ENTITY
  name
  address
  phone_number

client
isa: person
  account
  goal(account,identify_client)
  credit_margin
  goal(credit_margin,protect_bank_interest)
  safety_box
  necessary_condition:
    (client(Person) <-
       account(Person,Account))
    or
    (client(Person) <-
       safety_box(Person,Safety_box))

withdraw(Person,Amount)
isa: TRANSACTION
  precondition:
    account(Person,Account)
    goal(account,identify_client)
  procedure:
    debit(Account,Amount)
    issue_money(Person,Amount)
    goal(issue_money,inc_client_liquid)

deposit(Person,Amount)
isa: TRANSACTION
  precondition:
    account(Person,Account)
    goal(account,identify_client)
  procedure:
    receive_money(Person,Amount)
    goal(receive_money,dec_client_liquid)
    credit(Account,Amount)

debit(Account,Amount)
isa: ACTION
  precondition:
    balance(Account,Balance)
    goal(balance,protect_bank_interest)
    Balance > Amount
  procedure:
    sub_fm_balance(Account,Amount)
    goal(sub_fm_balance,record_transaction)

credit(Account,Amount)
isa: ACTION
  precondition: nil
  procedure:
    add_to_balance(Account,Amount)
    goal(add_to_balance,record_transaction)
```

Figure 1. Frames of the domain theory for banking

The partial explanation produced for the training example contains the unproven antecedent `account(bob,Account)` as shown in the explanation tree of figure 3. The partial explanation generated can be changed into a plausible explanation by using abduction to transform `account(bob,Account)` as an hypothesis for the training example feature `client(bob)`.

```
The training example is:
 withdraw(bob,100)

The facts are:
 client(bob)
 address(bob,101_Colonel_by)
 balance(acc_1,150)
 sub_fm_balance(acc_1,100)
 issue_money(bob,100)
```

Figure 2. The training example for the abduction scenario

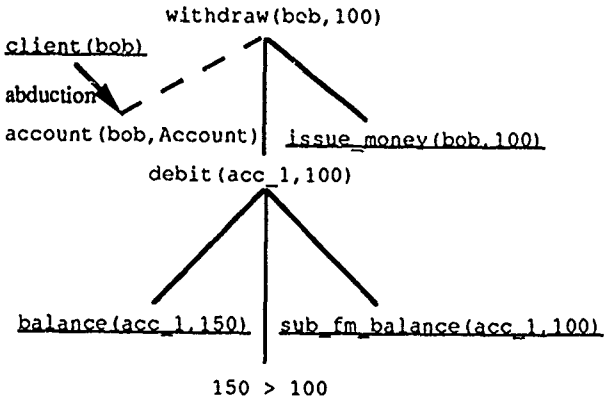


Figure 3. An example of abduction

As usual in EBL, irrelevant features of the training example, e.g. `address(bob,101_Colonel_By)`, are left out of the explanation.

3.3 Examples Of Analogical Reasoning

Analogical reasoning is used to build a plausible explanation re-using parts of one or several partial explanations and replacing the unproven antecedents by training example features sharing the same goal. The parts re-used correspond to the antecedents of partial explanations found as facts in the training example.

Two examples of analogical reasoning will be presented. The first example shows how a plausible explanation can be obtained using a partial explanation and replacing its unproven antecedents by analogous training example facts. The second example will show how multiple partial explanations can be combined to construct a plausible explanation.

3.3.1 Example 1: The Transaction `borrow`.

This process will be illustrated using the example of the transaction `borrow(bob,1000)`. Figure 4 shows the training example.

The domain theory is incomplete because the specification of the transaction `borrow` required to explain the training example is missing. Consequently, a set of partial explanations will be produced. Figure 5a and 5b illustrate the partial explanations that were obtained using transactions `withdraw(Person,Amount)` and `deposit(Person,Amount)`.

```
The training example is: borrow(bob,1000).

The facts are:
  account(bob,acc_1)
  credit_margin(bob,3500)
  record_loan(bob,1000)
  issue_money(bob,1000)
  car(bob,car_of_bob)
  value(car_of_bob,12000)
```

Fig. 4. The training example for `borrow(Person,Amount)`

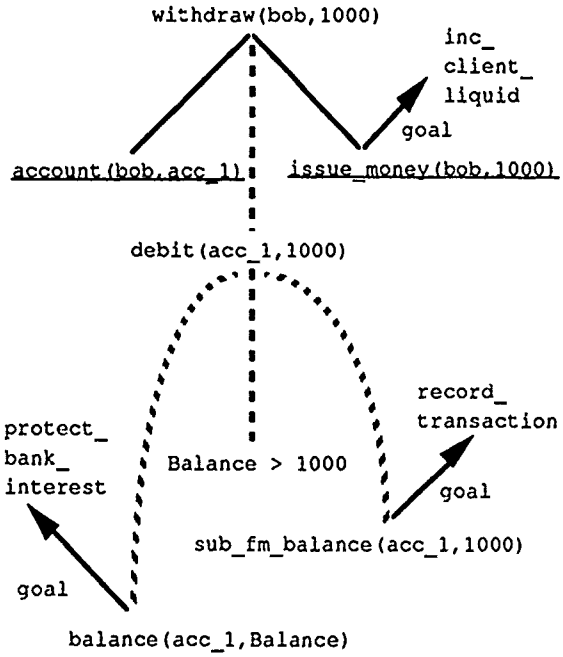


Figure 5a. Partial explanation for `borrow` produced using `withdraw`

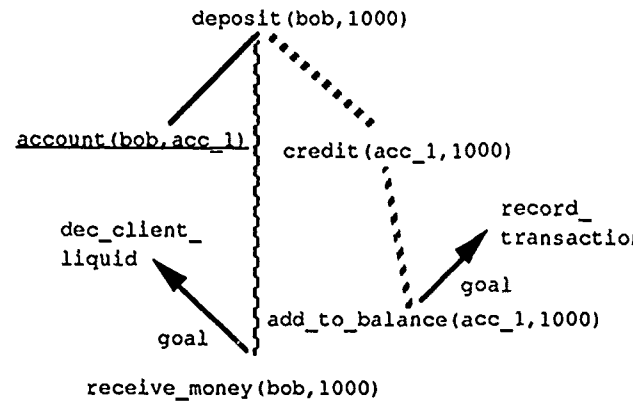


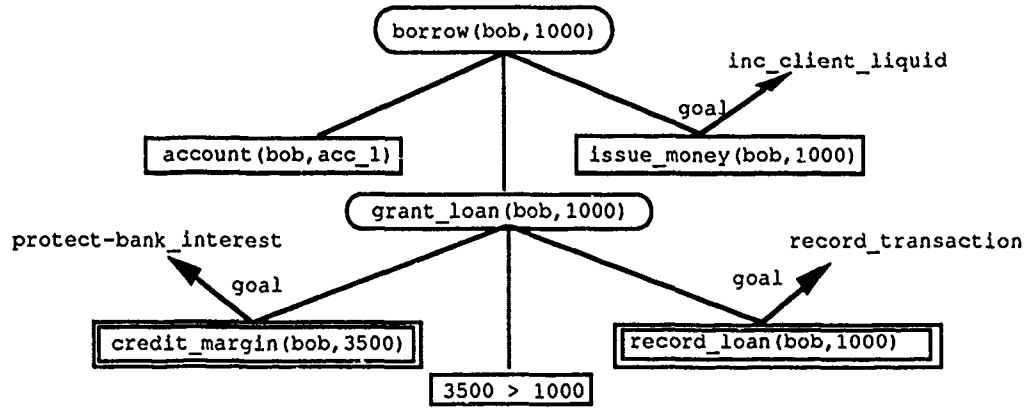
Figure 5b. Partial explanation for `borrow` produced using `deposit`

The dashed lines in the partial explanations indicate which antecedents were left unproven. The underlined antecedents in the partial explanations indicate which antecedents were found as training example features. The heuristic was used to score each partial explanation. The partial explanation built using `withdraw(Person,Amount)` scored the highest since it contains more underlined antecedents than the partial explanation built using `deposit(Person,Amount)`.

Figure 6 shows that the proven antecedents of withdraw are re-used in the plausible explanation of `borrow`. The unproven antecedents were replaced by selected training example features. A training example feature is selected to replace an unproven antecedent if it shares the same goal. In the borrow example, `account(bob,acc_1)`, `issue_money(bob,100)` and the operator ">" were all re-used. The antecedent `balance(Account,Balance)` was replaced by the training example feature `credit_margin(bob,3500)` because they have the same goal: protecting the bank interest. Similarly, but for a different goal, `sub_fm_balance(Balance,Amount)` was replaced by `record_loan(bob,1000)`. LISE asked the user for a name to replace `debit(Account,Amount)` in the plausible explanation. The user provided the name `grant_loan`.

Training example features that were not re-used nor selected are deemed irrelevant. In the example, `car(bob,car_of_bob)` and `value(car_of_bob,12000)` are irrelevant.

Initially, a set of partial explanations was obtained because the domain theory did not contain the frames for `borrow` and for `grant_loan`. To circumvent the incompleteness problem, a plausible explanation was built from partial explanations. The next step is to synthesize the missing frames from the plausible explanation and to insert them in the domain theory. For



Legend:

Feature of partial explanation re-used

Selected training example features replace unproven antecedent since they share the same goal

Name generated by the user

Figure 6. The plausible explanation for `borrow` produced using `withdraw`

that task, the structure of the frames of the partial explanation is used as a guide to create the new frames from the plausible explanation. The structure of the frame of `withdraw` is used to create the frame of `borrow` and the structure of the frame of `debit` is used to build the frame of `grant_loan`. The frames are shown on figure 7.

```
Name: grant_loan(Person,Amount)
 isa: ACTION
  precondition:
  credit_margin(Person,Margin)
    Margin > Amount
  procedure:
    record_loan(Person,Amount)
Name: borrow(Person,Amount)
 isa: TRANSACTION
  precondition:
    account(Person,Account)
  procedure:
    grant_loan(Person,Amount)
    issue_money(Person,Amount)
```

Figure 7. The frames for `grant_loan` and for `borrow`

3.3.2 Example 2: The Transaction `transfer`

A single partial explanation might contribute to explain several features of the training example while leaving out other relevant ones. Such a situation can be suspected when several partial explanations match different features of the training example. Figure 8 pictures the training example `transfer(Person,Amount)` which will be learned using two partial explanations.

The domain theory is incomplete since it does not include the frame of `transfer`. Consequently, partial explanations are produced (figure 9a and 9b). It is interesting to note that the partial explanation obtained using `withdraw` covers some training example features (the underlined antecedents) while the one obtained using `deposit` covers other features. In that case, we recognize that a single partial explanation will not provide enough foundation to build the plausible explanation. Both partial explanations will be integrated to produce the plausible explanation of `transfer` (figure 10).

```
The training example is: transfer(bob,100).

The facts are:
   account(bob,acc_1)
   account(bob,acc_2)
   phone(bob,992-2318)
   balance(acc_1,350)
   sub_fm_balance(acc_1,100)
   add_to_balance(acc_2,100)
```

Figure 8. The training example for the transaction `transfer(Person,Amount)`.

The unproven antecedents `issue_money` of `withdraw`, and `receive_money` of `deposit`, do not raise any problem since the goal hierarchy indicates that they can be mutually removed when the `person` and the `amount` are the same.

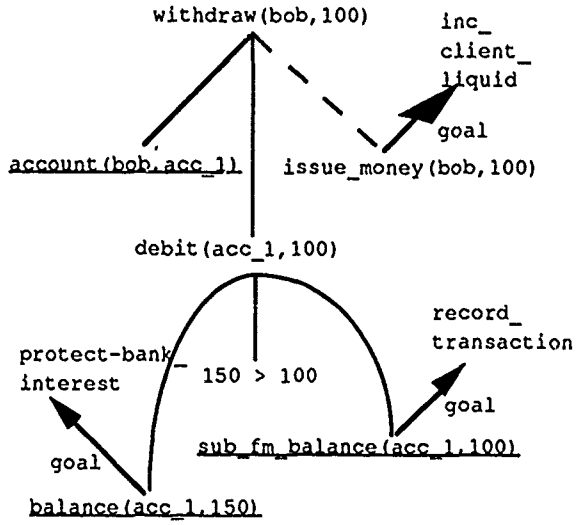


Figure 9a. Partial explanation for `transfer` produced using `withdraw`

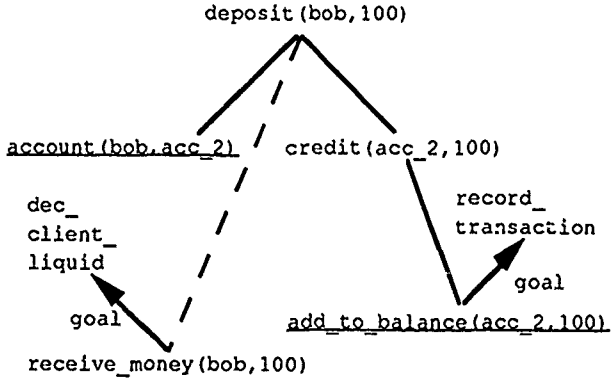


Figure 9b. Partial explanation for `transfer` produced using `deposit`

After the plausible explanation was built, the frame (figure 12) for `transfer` is synthesized and integrated with the domain theory. It is the only frame added to the domain theory since the frames of `debit` and `credit` already exists.

4 Conclusion

This paper has presented a learning method in which EBL is used in concert with an incomplete domain theory. The approach to deal with the incomplete theory is by integrating abduction, analogical reasoning and case-based reasoning. Inasmuch as our system augments the deductive closure of its domain theory by adding rules into it, it achieves knowledge-level learning. This is seldom the case for EBL systems.

[Ellman 1989] mentions that the effort in EBL research addresses the problems of justified generalization, chunking, operationalization and justified analogy. With regard to that classification, our work addresses the problem of operationalization since our objective is to translate a non-operational expression (i.e. user requirement) into an operational one (i.e. a specification).

[Ellman 1989] divides the methods to handle the incomplete domain theories into the analytical methods and the empirical methods. The usage of abduction, analogical reasoning and case-based reasoning categorizes LISE as an analytical methods. Other analytical methods require that a pair of training examples with similar functions be presented simultaneously [Hall 1988] or they need an experimentation theory to refine the domain theory [Rajamoney 1988].

Empirical methods deal with incomplete domain theories (e.g. [Pazzani 1988],[Fawcett 1989]) by conjecturing rules to fill holes in the partial explanation and, using subsequent training examples, empirically refine the conjectured rules.

Building partial explanations in EBL can be very expensive since the entire domain theory must be examined. Our system was provided with a heuristic to limit the search for partial explanations. The heuristic is to require that the order of features in the training examples be strictly equivalent to the order of the antecedents in the rules. The same heuristic is also employed in our case-based reasoning system.

The following is an analysis of the cost concurred by our approach. The search of EBL produces a list of `P` partial explanations for a training example composed of `N` facts. To assign the score to the partial explanation and to rank them, using sorting, brought a cost of `O(P log P)`. Considering that there is `M` unproven antecedents in the best partial explanation and that there is `R` rules having each unproven antecedents in their right-hand side, we obtain the cost of `R*M` for the search related to abduction. If the abduction fails, this cost will be augmented by `N*M` which is needed to solve the training example using analogical reasoning. Thus, the cost for the first two steps of of our approach is `O(P log P + (R+N) * M)`. A good domain theory will provide partial explanations with small values for `M`.

The cost will increase more considerably if case-based reasoning is applied to the training example. The cost of matching cases in our approach will be polynomial since the features are ordered in our cases and in the training examples. The cost of assigning the score and ranking the cases is `O(C log C)`, where `C` is the number of cases having at least one match. The cost of adapting cases is `S*N` where `S` is the number of unmatched case features and `N` is the number of training example facts. The ordering imposed on the facts of the training examples,

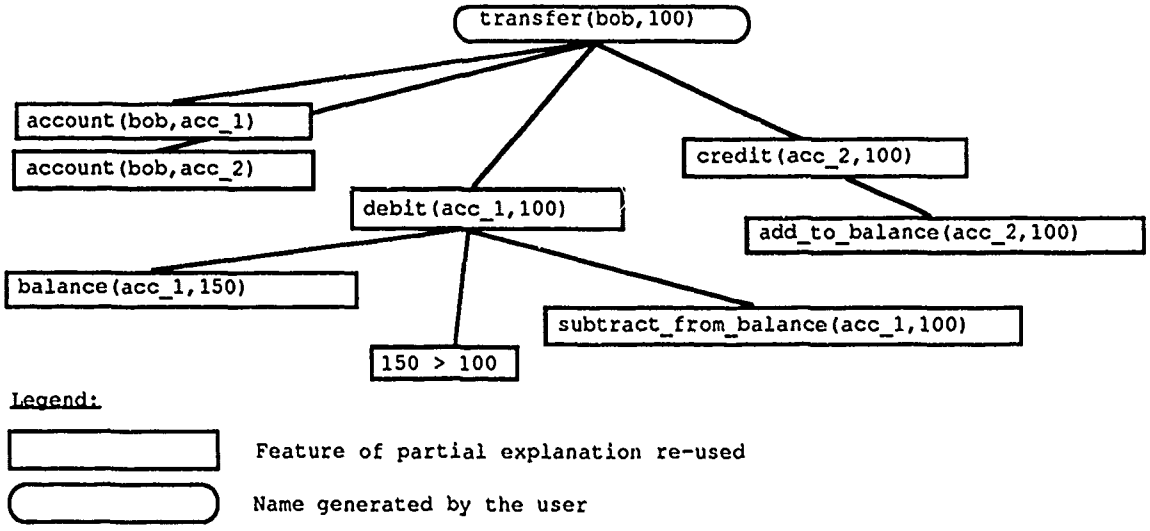


Figure 10. The plausible explanation for `transfer` produced using `withdraw` (the goals are omitted for clarity)

```
Name: transfer(Person,Amount)
 isa: [transaction(Person)]
   precondition:
     account(Person,Account1)
     account(Person,Account2)
   procedure:
     debit(Account1,Amount)
     credit(Account2,Amount)
```

Figure 11. The new frame for `transfer` added to the domain theory.

on the predicates of the rules, and on the features of the cases ensued a polynomial cost rather than the prohibitive NP-complete problem of matching unordered sets of features.

The cost of abduction and analogical reasoning being of the same order, we chose to apply abduction first in our approach because abduction does not change the domain theory while analogical reasoning does. We believe that the domain theory should be modified conservatively, only when abduction can not transform a partial explanation into a plausible one.

LISE was successfully implemented in Prolog. A specification for a small banking system was developed. We are currently working on the specification of a fleet management system

5. Future Work

A future goal of our research is to study the validity of our approach from a cognitive science viewpoint. We are interested in the way people select partial explanations when faced with problems and how people apply analogical reasoning to create new explanations from previous ones. We hope to obtain an answer to these questions from a current experiment designed jointly with a cognitive scientist.

As for any non-empirical learning method, learning is limited by the amount of initial knowledge contained in the domain theory and the case-base. A future goal for this research is to make the system less dependent on the initial knowledge.

Acknowledgement

The research described here was done at the Knowledge Acquisition Laboratory in the Department of Computer Science, University of Ottawa. The work of the Laboratory is sponsored by Natural Sciences and Engineering Research Council of Canada, the Ontario University Research Incentive Fund, Cognos, Inc., and the University of Ottawa. Financial support for the first author was provided by the Department of National Defence.

References

Brodie, M.L. and Ridjanovic, D. (1984). "On The Design and Specification of Database Transactions," in *On Conceptual Modelling (Perspectives from Artificial Intelligence and Programming Languages)*, M. L. Brodie, J. Mylopoulos and J. W. Schmidt, ed. , Springer-Verlag, New York, pp. 277-312, 1984.

DeJong, G.F. and Mooney, R. (1986). "Explanation-Based Learning: An Alternative View, " *Machine Learning*, vol. 1, pp. 145-176, 1986.

Ellman, T. (1989). "Explanation-Based Learning: A survey of Programs and Perspective," *ACM Computing Surveys*, vol. 21, no. 2, pp. 163-221, 1989.

Fawcett, T. (1989) "Learning from Plausible Explanations, " Procs. of *9th International Conference on Machine Learning*, Ithaca, pp. 37-39, 1989.

Genest, J. and Matwin, S. (1990) "Building Systems Specification using Explanation-Based Learning with Incomplete Theory", Research Report, University of Ottawa, TR-90-01, 1990

Hall, R.J. (1988). "Learning By Failing to Explain: Using Partial Explanations to Learn in Incomplete or Intractable Domains, " *Machine Learning*, vol. 3, no. 1, pp. 45-77, 1988.

Holte, R.C., Walker, L.E. and Porter.B.W. (1989) "Concept Learning and the Problem of Small Disjuncts, " Procs. of *IJCAI*, Detroit, MI, 1989.

Matwin, S. and Plante, B. (1990) "Learning of Flexible Concepts with Theory Revision", Research Report, TR-90-02, 1990

Mitchell, T., Keller, R.M. and Kedar-Cabelli, S.T. (1986). "Explanation-Based Generalization: A Unifying View, " *Machine Learning*, vol. 1, pp. 47-80, 1986.

Pazzani, M. (1988) "Integrated Learning with Incorrect and Incomplete Theories, " Procs. of *5th National Conference on Artificial Intelligence*, pp. 555-559, 1988.

Pople, H.E., Jr. (1973) "On the Mechanization of Abductive Logic, " Procs. of *3rd IJCAI*, 1973.

Rajamoney, S.A. (1988) "Experimentation-Based Theory Revision, " Procs. of *AIII Spring Symposium on Explanation-Based Learning*, pp. 7-11, 1988.

Using Abductive Recovery of Failed Proofs for Problem Solving by Analogy

Yves Kodratoff
CNRS & Université Paris Sud, LRI, Bldg 490, 91405 Orsay, France
George Mason University, AI Center, Fairfax, Virginia 22030, USA

Abstract

We propose a solution to problem solving by analogy which is an alternative to Carbonell's transformational analogy. Given a plan that succeeds for the base, we apply the plan to the target and propose to correct its failures by an abductive recovery mechanism inspired from abductive recovery from failed proofs.

1 Introduction

The analogy scheme we shall use in this paper is quite a classical one (Winston, 1982; Gentner, 1983; Chouraqui, 1985; Falkenhainer, Forbus, and Gentner, 1986; Carbonell, 1983, 1986; Kedar-Cabelli, 1988; Kodratoff, 1988). It can be described as follows. Let us suppose that we dispose of a piece of information, the **base**, that can be put into the form of a doublet (A, B) in which it is known that B depends on A. This dependency will often be causal, and it does not need to be very formal nor strict. In the following, we shall call this relation β, and refer to it as the **causality**[1] of the analogy. Suppose now that we find an other piece of information, the **target**, (A', B') that can be put into the same form, and such that there exists some resemblance (similarity) between A and A'. In the following, we shall call this relation α, and refer to it as the **similarity** of the analogy. Let us call β' the causal dependency between A' and B', and α' the similarity between B and B', as shown in the figure below.

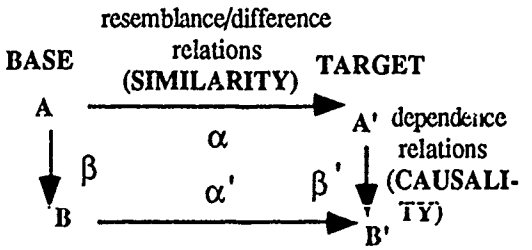


Figure 1: The general scheme of analogy

[1] It is also often called the *internal dependency* of the base. Arguments for calling it *causality* are found in section 2 and in Kodratoff (1990).

The very first idea one can try in applying this scheme to problem solving is to consider that A is the initial state of the system, B is its final state, and β is a sequence of applications of operators leading from A to B. The analogy problem then becomes to find a β' allowing to go from a new initial state, A', similar to A, to a new final state B'. This view is summarized in figure 2a below.

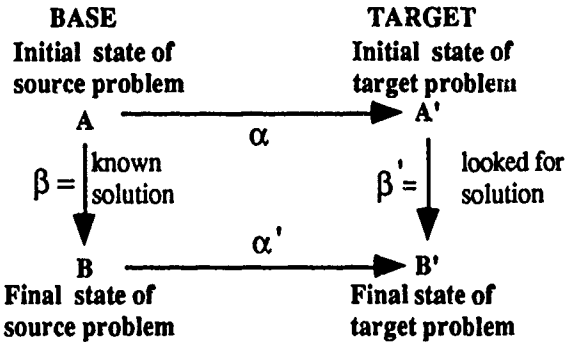


Figure 2a: A classical view of the use of analogy for problem solving.

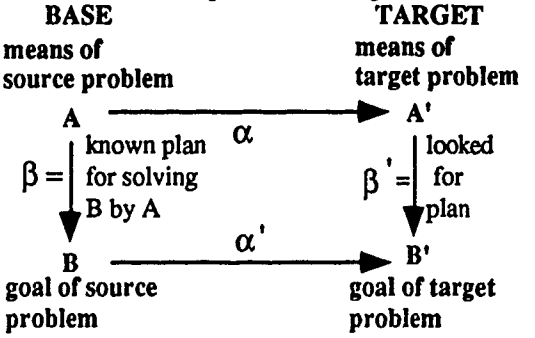


Figure 2b: Our scheme for using analogy in problem solving. It requires a plan for a solution, instead of a particular solution as in figure 2a.

In this paper, we would like to present a somewhat different view of problem solving, in which A is the set of means of the source problem, and B the set of its goals, and β is a plan for going from A to B. This view is summarized in figure 2b, above. The means of the problem are made of all the knowledge necessary to solve the

problem, and of the instantiations particular to the problem at hand. The goals of the problem is the set of the consequences, interesting for solving the problem at hand, of applying plan β to A. In the following we shall elaborate an example coming from (DeJong and Mooney, 1986). The means of the base are *abducting a rich person's child*, and the goals be *kidnapper becomes rich*. We are given a plan (the one of DeJong and Mooney, 1986) that solves the problem of achieving goal B by means A. This plan gives the cause why it is possible to achieve goal B by means A, it will therefore be seen here as the causality β linking A and B. We are also given an other problem, i.e., goals B' and means A' (e.g., A' may be: *abducting a famous politician*, and B' may be: *terrorists advertise their political cause*). The analogical problem is then to use the plan β in order to invent a new plan β' that will achieve B' by A' (e.g., *how terrorists can get advertisement by abducting a famous politician?*).
Our proposal for finding which transformations to apply to plan in order to obtain plan β' is precisely to attempt applying plan β to A', analyze the partial successes and failures of this application, and induce from them a β' that will include the successes and eliminate the failures. Thus, recovering from a failed analogy is central in our view of problem solving by analogy.
We shall present here a scheme which is very near to abductive recovery of proof failures as presented by Cox and Pietrzykowski (1986), Duval and Kodratoff (1989). A detailed example of abductive recovery is presented in section 6. The fundamental technique implementing such a recovery system is the inversion of resolution as described in (Muggleton and Buntine, 1988; Rouveirol and Puget, 1989). Let us now see what these techniques are precisely, and how they can be applied to recovery from plan failures.

2 Causal knowledge in creative analogies

In the case of recognition and evaluation of existing analogies, there are no needs to draw a difference between similarity and causality. In that case, causality is just one more similarity between source and target. On the contrary, causality is central to the generation of new analogies.
As an illustration, consider the following analogy, proposed in (Russell, 1989).
From *nationality*(Louis, France) & *nationality* (Antoinette, France) & *native_language*(Louis, French), Russel (1989) finds by analogy that *native_language*(Antoinette, French). Adding new information about Louis and Antoinette (we assume here that these characters are the royal couple sent to the guillotine during the French revolution, thus taking Antoinette for Marie-Antoinette), like *lives_in*(Louis, France) and *lives_in*(Antoinette, France) will increase the similarity between Louis and Antoinette, therefore increasing the relevance of the analogy. Conversely, adding information like *male*(Louis) & *born_in*(Louis, France) and *female*(Antoinette) & *born_in*(Antoinette, Austria) will decrease its relevance. With this added information, the analogy can be written without causality, as follows.

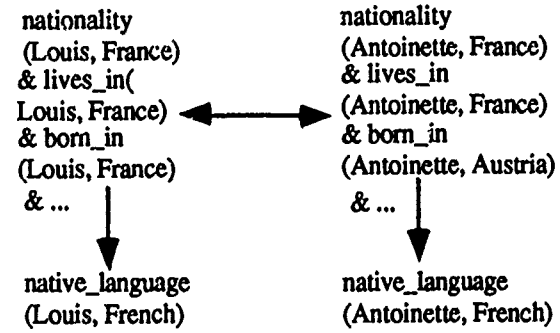


Figure 3. The given analogy, without causality.

On the contrary, one can also consider that some of this information is causal. It will allow us to find back the given analogy when one considers that β_1 = *lives_in*(Louis, France) and β'_1 = *lives_in*(Antoinette, France) as causalities for the fact of being native French speaker, and when one does not take into account that Antoinette is born in Austria.

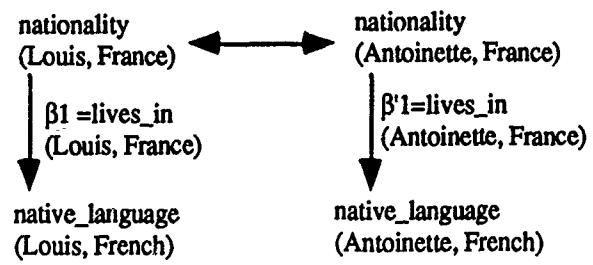


Figure 4. Inventing again the given analogy by using a causality of the form "x lives_in y" in order to explain that "native_language(x) = language(y)".

Consider now that one adds the following information about Louis: *born_in*(Louis, France). Then, the similar information about Antoinette, β'_2 = *born_in*(Antoinette, Austria), leads to the analogy *native_language* (Antoinette, German), or to *fluent_in* (Antoinette, French), depending on the causality to be used. If β_2 and β'_2 are considered as causal and β_1 and β'_1 are considered as factual, then the analogy should give *native_language*(Antoinette, German).
In this analogy, one is implicitly using theorems of the kind: $\forall x$ [*born_in*(x, France) $\Rightarrow$ *native_language*(x, French)] and $\forall x$ [*born_in*(x, Austria) $\Rightarrow$ *native_language*(x, German)].

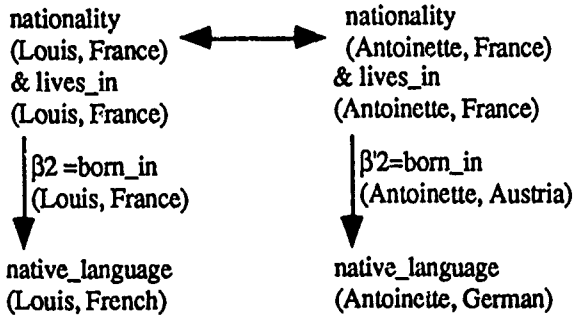


Figure 5. Inventing another analogy by using a causality of the form "x born_in y" in order to explain "native_language(x) = language_of(y)".

The choice of using these theorems follows from the choice of causality. Let us show why in three steps.
First step: Understanding causality. In the present case, the causality is β_2 = *born_in*(Louis, France), which "explains" why *native_language*(Louis, French). From this, we can infer that the analogy has been using ways of deducing the result from its causality. Therefore, we have to consider theorems that have a generalization of *born_in*(Louis, France) in their premise, and that have a generalization of *native_language*(Louis, French) in their conclusion. In other words, we have to consider the different ways by which one might prove something of the form *native_language*(x, y) from something of the form *born_in*(x, y). This may be very difficult, and the difficulty of finding the link between the causality and its consequences may become a huge task by itself. In the very case we are looking at presently, this inference can be done in a single step by using the theorem $\forall x$ [*born_in*(x, France) $\Rightarrow$ *native_language*(x, French)].
Second step: Using similarity. Similarity tells us that Louis in the base must be replaced by Antoinette in the target. Therefore, we guess that the causality in the target is β'_2 = *born_in*(Antoinette, Austria).
Third step: Combining causality and similarity. We look for theorems the premise of which is a generalization of *born_in*(Antoinette, Austria), and the conclusion of which is a generalization of *native_language*(x, y). Once more, this step may be very complicated but, in this case, we find in one step that $\forall x$ [*born_in*(x, Austria) $\Rightarrow$ *native_language*(x, German)] is the looked for theorem. Applying it to the premise *born_in*(Antoinette, Austria) leads to the conclusion *native_language*(Antoinette, German), which becomes the conclusion of our analogy, as shown in figure 5.

Consider now that β_1 and β'_1 are causal and that β_2 and β'_2 are factual. Then, the analogy should give *fluent_in*(Antoinette, French).

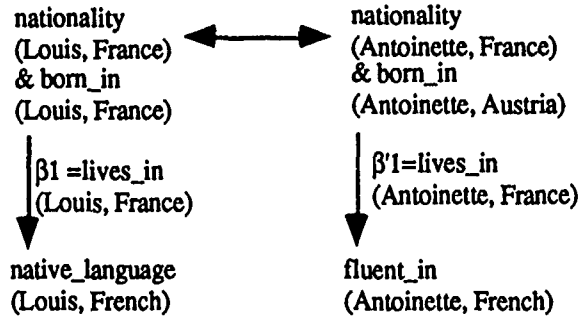


Figure 6. Inventing another analogy by using a causality of the form "x lives_in y". In the context of "born_in(x, y) this explains "native_language(x) = language_of(y)". In the context of "NOT born_in(x, y), this explains "fluent_in(x, language_of(y))".

In this analogy, the theorems implicitly used are: $\forall x$ [*lives_in*(x, France) & *born_in*(x, France) $\Rightarrow$ *native_language*(x, French)], $\forall x$ [*lives_in*(x, France) & $\neg$*born_in*(x, France) $\Rightarrow$ *fluent_in*(x, French)]. Applying the above three steps in the same way would lead us to choose these theorems (instead of the two above). In other words, we can say that we have been using theorems the left-hand side of which can be instantiated by *lives_in*(Antoinette, France) , such as $\forall x$ [*lives_in*(x, France) & .. $\Rightarrow$ ..].

When creating analogies, the choice of an information as causality will orientate the invention process. on the contrary, when analyzing existing analogies, some informations are more relevant all kinds of information play a role in rating the given analogy. For instance, in the case of the given analogy above, one might well use both informations *lives_in*(Louis, France) and *born_in*(Louis, France) to rate the given analogy. On the contrary, when creating analogies, one has to choose between the available informations which one is of causal nature, and this choice changes the output of the analogy process. In other words, from the analysis point of view, *native_language*(Antoinette, German) is as good an analogy as *fluent_in*(Antoinette, French), while from the invention point of view, they differ in the information that has been chosen as causal. In practise, one should always dispose of large amounts of theorems such as those exemplified in example 1, possibly even of theorems that contradict each other. Analogy, which contains for us a choice of causality, allows to choose which to use.

3 Inversion of resolution

Let us start with an initial theory T. Suppose that T meets an example E that it cannot explain, because E cannot be derived from T. In this case, one solution is to consider that T is incomplete. Induction then amounts to

find a new theory T' that allows to derive both the initial theory T and the example E.
Inversion of resolution is based on three basic operators, called absorption, intraconstruction and truncation (Muggleton and Buntine, 1988; Rouveirol and Puget, 1989). Let us illustrate how inversion of resolution works by the following theory defining family relationships. Since inversion of resolution has been developing up to now in PROLOG, we shall describe these examples in PROLOG notation.

T_1: grandfather(X,Z) :- father(X,Y), father(Y,Z).
T_2: father(X,Y) :- child_of(Y,X), sex(X,male).
T_3: mother(X,Y) :- child_of(Y,X), sex(X,female).

Suppose now that the following example is met.

E: grandfather(tom,liz):- father(tom,helen), child_of(liz, helen), sex(helen,female).

It is clearly not entailed by the available theory. Let us show how inversion of resolution allows to induce form T and E a new theory, T', that entails T and E.

3.1 Absorption

Absorption of the clause T_i of the theory by the example E is possible if the body of the clause T_i can be unified with a part of the body of E. In the example, only one clause of the theory can be absorbed by the example, T_3 because its body *child_of(Y,X), sex(X, female)* can be unified with the part of E *child_of(liz, helen), sex(helen, female)* with the substitution {Y/helen, X/liz}
Absorption then replaces the body of the absorbed clause T_i by its head properly substituted in the body of the absorbant clause E. This gives a new form to the example.

E': grandfather(tom,liz) :- father(tom, helen), mother (helen, liz).

3.2 Intraconstruction

It can occur between two clauses C_1 and C_2 if the heads of C_1 and C_2 match, and their bodies match partially. It proceeds in three steps.

Firstly, it generates a new clause $T_{1a_{tempo}}$ the head of which is the least generalization of the heads of C_1 and C_2. Its body is the least generalization of common parts of the bodies of C_1 and C_2. For instance, E' and T_1 can undergo the first step of intraconstruction, giving the following clause.

$T_{1a_{tempo}}$: grand-father(U,W):- father(U,V).

The second step takes care of the left-over literals in C_1 and C_2. Here, the literal *mother(helen,liz)* of E' and and *father(Y,Z)* of T_1 have been left over during the first step of intraconstruction. Intraconstruction then introduces a new literal, arbitrarily called *newp* that becomes the head of these left-over. The arguments of *newp* have to be carefully chosen to keep the variables bindings that were present in C_1 and C_2. Note that the two clauses thus built define the predicate *newp* in extension, i.e., no generalization has been occurring. In our example, these new clauses are

T_{1b} : newp(Y,Z) :- father(Y,Z).
$T_{1c_{tempo}}$: newp(helen,liz) :- mother(helen,liz).

The third step of intraconstruction generates a new version of the clause of the theory which has been undergoing intraconstruction (here, T_1), by replacing the left-over part of this clause by the new predicate newp, taking again care of introducing the correct variable binding. This generates the clause

T_{1a}: grand-father(U,W) :- father(U,V),newp(V,W).

3.3 Truncation

The truncation operator is a generalization operator which must be controlled in some way. In our example, it replaces constants by variables in order to give the same degree of generality to the clauses generated by intraconstruction. Applying truncation to $T_{1c_{tempo}}$ gives the more general clause

T_{1c}: newp(Y,Z) :- mother(Y,Z).

4 Formation of a new theory

The new theory T' is formed by deleting from T the original clause that underwent intraconstruction, and by adding to T the clauses generated by intraconstruction and truncation. In our example, this gives the set of clauses (T_{1a}, T_{1b}, T_{1c}, T_2, T_3). T' is able to recognize the new example, and all other examples of maternal grandfathers. It is also possible to consult an oracle, who might propose to call *newp* by the name *parent*. This is useful for the sake of knowledge base readability.

We consider here the case where the theory has already been used at least once with success. We shall use this positive past experience in order to drive the inversion of resolution. For example, in problem solving by analogy, the base case is such a success. Suppose now, that new problems are given to the system, and that it is unable to solve them. Similarly, a theory can be able or not to "recognize" an example as belonging to the theory. If it fails to do so, we can consider that the reason of the failure is the incompleteness of our theory, and we will accordingly attempt to make it complete. We propose here to increase a theory by two different abduction mechanisms.
Suppose that we start with a theory Th_0 and an example E_0 of a concept C, such that E_0 can be proven from Th_0. This proof generates a complete proof tree Pc_0. Using now classical EBG techniques (Mitchell et al., 1986), this proof tree can be generalized enough to keep the proof sufficiency to cover the example. Suppose also that we meet now another example E_1 of C, such that it is not recognized by Th_0. In that case, we will suppose here

that, nevertheless, a partial proof tree Pp_1 can be generated for E_1. Our solution to abduction consists in proposing abduction mechanisms that will make complete Pp_1, thus obtaining a complete proof Pc_1. Our rule to control abduction is to try to obtain a Pc_1 which as "close" as possible from P_0. When a Pc_1 has been obtained, inversion of resolution allows to complete the theory accordingly.

The following example inspired from DeJong and Mooney (1986) will be used as an illustration. In this example, our aim is learning a definition of the concept of suicide Kill(x,x). The domain theory Th_0 contains the following rules

Theory Th_0

kill(A,B) :- hate(A,B), possess (A,C), shot-gun(C).
hate(W,W) :- depressed(W).
possess(U,V) :- buy(U,V).

where A, B, C, U, V, W, Z are variables.
The training instance E_0 is a suicide, described by the following facts.

E_0

depressed(john).
buy(john,obj_1).
shot-gun(obj_1).
:- kill(john,john).

Let us call Pc_0 the proof that E_0 is a consequence of the theory Th_0. Its proof tree can be generalized as follows (DeJong and Mooney, 1986).

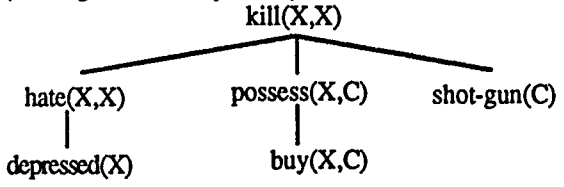


Figure 7. Generalized proof that John has been committing suicide.

Suppose now that the system is provided with an example E_1 of concept of suicide which is not recognized by the theory. Supposing that a partial proof Pp_1 of E_1 can be obtained, we will try to complete each of these Pp_1 in such a way that it becomes are as "close" as possible of Pc_0.

As an illustration of such a E_1, consider an other suicide instance TR_1 described by the following facts.

E_1

depressed(mary).
buy(mary,obj_1).
sleeping-pills(obj_1).
price(obj_1, 6).

where obj_1 is a constant.
We will be unable to prove kill(mary,mary) because she has no shot-gun. Nevertheless we obtain one partial proof, and only one in this case.

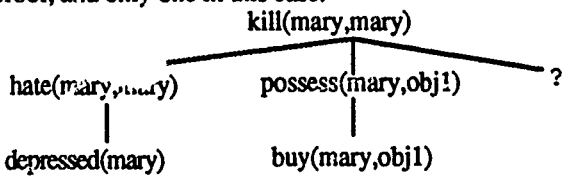


Figure 8. Pp_1: Partial explanation of Mary's suicide.

Pp_1 obviously matches a sub-tree of Pc_0.

First induction (abduction the missing part)

We attempt to complete Pp_1 by viewing Pc_1 as an instance of Pc_0, this is one of the possible definitions of "closeness". Therefore, Pp_1 will be completed by taking the missing pieces from Pc_0, appropriately instantiated.
In our example, such a forced matching leads to Pc_1, in which the missing part of Pp_1 has been replaced by shot-gun(obj_1). In this first abduction, the cause of the failure is attributed to our supposed "ignorance" that obj_1 (i.e. sleeping-pills) is actually a shot-gun. This mechanism has already been considered in other works about abduction such as Cox and Pietrzykowski (1986). Our example shows that it can be quite a dangerous step to do since it leads to complete the theory by adding the "fact"
shot-gun(obj_1).
amounting to state that sleeping-pills are kinds of shot-guns.
In the view of using this abduction in an analogical process, it will be quite easy to check if this abduction allows to complete the solution of the target problem. If it does not, we propose to use another kind of induction, in which Pc_0 and Pc_1 are not supposed to match.

Second induction (induction of the missing theorem)

In this case, we try to add a new rule that will allow to complete the proof. One easily understand why this mechanism has not been taken into account so far; in principle one can add so many ridiculous rules that this approach seems to be hopeless.
For instance, in our example adding to Th_0 the rule
kill(X,X) :- sleeping-pills(C), price (C, 6)
will indeed allow to prove Mary's suicide. But it means that everyone will suicide when the price of sleeping pills reaches the value 6, which is totally irrelevant to the preceding suicide case.
In order to avoid adding such ridiculous rules, we define a new notion of distance between Pc_0 and Pc_0. Given two possible completed proof trees, Pc_1 and Pc_1', we shall collect the mismatches between Pc_0 and Pc_1, on the one

hand, and between Pc_0 and Pc_1' on the other hand. We shall say that Pc_1 is closer to Pc_0 than Pc_1' if the number of mismatches between Pc_1 and Pc_0 is less than the number of mismatches between Pc_1' and Pc_0, with an exception for zero mismatches that would drive us back to the first kind of induction. In other words, we consider the least mismatch, a complete matching being already covered by the first abduction.When they are equal, we shall say that Pc_1 is closer to Pc_0 than Pc_1' when the conceptual distance (supposedly defined) between the mismatches is less for Pc_1 than for Pc_1'.

In our example, it is clear that the number of mismatches between Pc_0 and the proof tree obtained by using

kill(X,X) :- sleeping-pills(C), price(C, 6)

to prove Mary's suicide is very high.

We shall rather try to use our knowledge about the objects possessed by Mary to complete Pp_1. For instance, completing it by sleeping-pills(obj_1) gives a proof tree Pc_1.

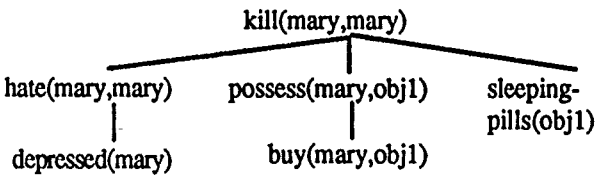


Figure 9. Pc_2. Sleeping-pills are viewed as the cause of Mary's death.

We can generalize this explanation by inversion of resolution. The two clauses

kill(X,X) :- depressed(X), buy(X,C), shot-gun (C)

kill(mary,mary) :- depressed(mary), buy(mary,obj_1), sleeping-pills(obj_1)

are generalized by intraconstruction and lead to the three clauses

kill(X,X) :- depressed(X), buy(X,C), new(C)

newp(C) :- shot-gun(C)

newp(C) :- sleeping-pills(C)

In this example, the values of newp(c) are a description in extension of the concept of "tool-for-suicide" which, actually, is a poorly defined concept. Besides guns and sleeping-pills, it covers also various cliffs, the Eiffel tower etc. If we would not have been driven by the first example, this abductive recovery would have been done with little caution. In other words, this kind of abductive recovery is hard to perform, but we claim that it is quite necessary, and, that it finds a justification within our frame.

5 Recovering from plan failures

Suppose that the plan β of the base problem amounts to the application of a sequence of operators $\{Op_1, ..., Op_n\}$. The means of the base problem contain a set of instantiations, called here σ, such that σ applied to $\{Op_1, ..., Op_n\}$ leads to fulfill the goals of the base problem. In other words, one has to prove that $\sigma\{Op_1, ..., Op_n\}$ does not contradict the goals of the base which amounts to proving that each Op_i is such that σOp_i does not contradict the goals of the base, and the post-conditions of σOp_i contain the pre-conditions of σOp_{i+1}.

The means of the target problem contain a set of instantiations, called here σ'. We propose to "compute" σ' $\{Op_1, ..., Op_n\}$ and to attempt proving that it does not contradict the goals of the target problem. Unless we are extremely lucky, this proof will fail. We then propose to apply the above abductive recovery techniques in order to generate a new sequence of operators $\{Op'_1, ..., Op'_p\}$ such that $\sigma'\{Op'_1, ..., Op'_n\}$ does not contradict the goals of the target problem.

Section 7 gives a detailed example on how to achieve such a recovery. In a few words, the general strategy we use is the following:

- recognize the parts of the proof that have been succeeding. If no success at all occur[2], we fail to recover.
- delete the operators that have been leading to a failure. This process introduces unknown values (i.e., variables) in the sub-sequence of $\{Op_1, ..., Op_n\}$ which is left. Call $\{Op"_1, ..., Op"_q\}$ this sub-sequence.
- prove that some instantiations $\sigma"$ coming from the knowledge base can insure that $\{Op"_1, ..., Op"_p\}$, together with these instantiations, does not contradict the goals of the target.
- verify that $\sigma"$ do not contradict σ'.
- Perform the union of the old σ' and of $\sigma"$.

This union together with $\{Op"_1, ..., Op"_r\}$ is a complete solution to the target problem.

6 Application to Analogy

In (Kodratoff, 1990), we analyze the ways similarity and causality can combine, and we define the concept of full analogies. We say that an analogy is **full** when the law of combination of causality and similarity is $\alpha \circ \beta \circ \alpha^{-1}$. The symbol $\circ$ represents the composition of the substitutions, i.e., the application of α^{-1} to A', then the application of β to the result of the last operation, and finally the application of α to this last result (many examples are given below). In this definition, we assume that $\alpha =$

[2] This is the case when none of the subproblems can be solved. For instance, in the case of Mary's suicide (section 4), suppose we are unable to prove also depressed(mary), buy(mary,obj_1). Then, our attempt to prove kill(mary,mary) is a complete failure during which we met no success at all.

α'. Let us now explain why this is not a real restriction to our scheme.
In our scheme, α is a set of replacements allowing to express how to transform A into A', and α' is a set of replacements allowing to express how to transform B into B'. It may well happen that α and α' do not concern to the same set of replacements. When this is the case, we shall always consider that "α = α'" is the global similarity, α ∪ α', between the whole base and the whole target. This hypothesis would restrict the generality of our scheme only if the replacements could be contradictory in some sense. In that case, the analogy would have to take into account some kind of contradiction within the base itself. The first work to be done when setting up a knowledge representation would then be to make explicit this contradiction, and by that, getting rid of it[3].
This definition can be represented as follows, together with our example.

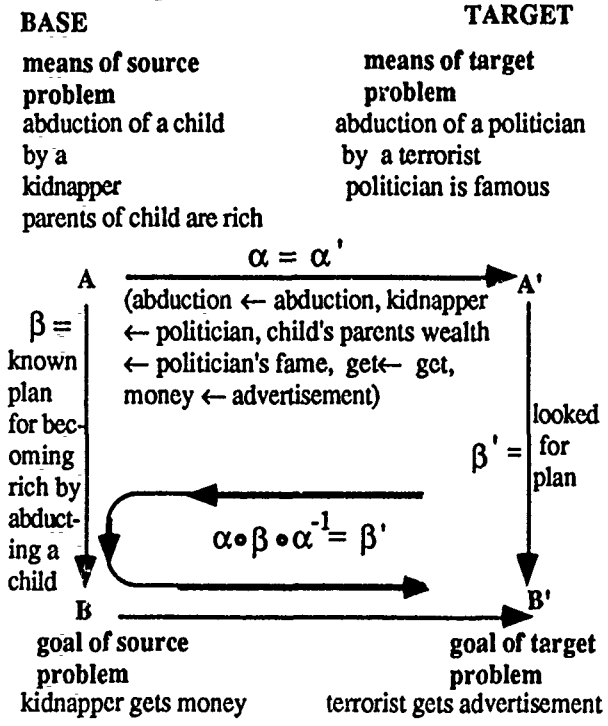


Figure 10. Applying our analogy scheme to the comparison of two kidnapping cases. The base case is the one of a child's kidnapping, the target case is the abduc-tion of a famous politician. The difficult substitution [child's parents wealth ← politician's fame] does not need to be found in advance to be able to apply analogy. Notice that the similarity is α ∪ α', it contains the substitutions allowing to go from one story to the other.

7 A detailed example of the generation of a new plan by abductive recovery of the failure of the old plan

We will illustrate this view of analogy by using the plans that have been learned by explanation-based learning (EBL) in (DeJong and Mooney, 1986) for kidnapping a rich person's child. Our aim will be to transform by analogy this plan in order to apply it to the case of terrorists abducting a famous politician in order to advertise their political cause.
The means and goals of the two problems are given in the scheme of figure 10 which summarizes the information contained in the two problems.
In their paper, DeJong and Mooney (1986) show that the abduction of a child can be represented by the application of the two operators of figure 11.

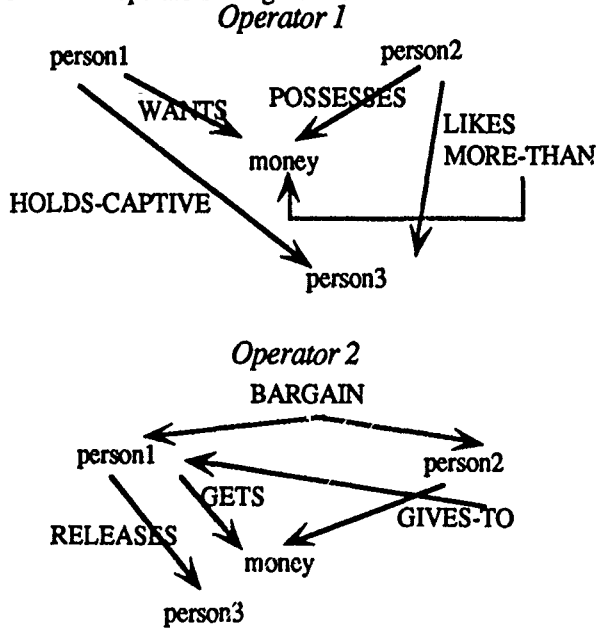


Figure 11. Generalized operators for kidnapping. The goal of kidnapping succeeds when the application of OP_1 and OP_2 succeeds. Each operation inside the operator can be also a problem by itself. For instance, how $person_1$ achieves to hold captive $person_2$ may be a problem by itself.

with the instantiations [$person_1$ ← kidnapper, $person_2$ ← child's parents, $person_3$ ← child, advertisement(abstract) ← advertisement(concrete)]. These operators are the "causality" β we have been introducing in our scheme. Actually, they do explain why the kidnapper can achieve his goal of getting money. Let us attempt a full analogy by computing $\beta \circ \alpha^{-1}$. In this case, we shall not use the

[3] This point could be made more precise, and be made quite formal. This is not our goal in this paper which is devoted to a more intuitive presentation. Here, it should be clear that as long as α and α' do not contradict each other, it will be simple to find a similarity that includes the two of them.

similarities as shown on figure 10, but the one which relevant to the application of β, i.e., kidnapper ← terrorist, ??? ← child's parents, politician ← child, money(concrete) ← advertisement(concrete)]. Where the ??? express the fact that we do not know yet who is going to play the role of the child's parents. This amounts to apply the instantiations: [person$_1$ ← terrorist, person$_2$ ← person$_2$, person$_3$ ← politician, advertisement(abstract) ← advertisement(concrete)] to OP$_1$ and OP$_2$. Applying these substitutions, we find operators that still contain the variable person$_2$. Our problem is now to use the background knowledge in order to find an instance of person$_2$ that will not introduce contradictions. Let us suppose that our background knowledge is represented by the following set of clauses.

1- WANTS(x, y, z) IF
NEEDS(x,z)

2- LIKES-MORE-THAN(x, y, money) IF
LOVES (x,y)

3- LIKES-MORE-THAN(x, y, money) IF
PARENT (x, y) & NOT-EXCEPTION-PARENTAL-RELATION(x, y)

4- LIKES-MORE-THAN(x, y, money) IF
STRONG-RELATION-BETWEEN (x, y)

5- EXCEPTION-PARENTAL-RELATION(x, y)
IF ...

6- HOLDS-CAPTIVE(x, y) IF ...

7- GIVES-TO(media, x, advertisement) IF
GIVES-TO(x, media, money)

8- GIVES-TO(media, x, advertisement) IF
GIVES-TO(x, media, exciting-news)

9- GIVES-TO(x, y, exciting-news) IF
EVENT(z) & KNOWS-OF(x, z) &
RELATIVE-TO(z, t) & FAMOUS(t)

10-MEDIA(x) IF
TV(x)

11-MEDIA(x) IF
RADIO(x)

12-MEDIA(x) IF
NEWSPAPER(x)

13- ...

Asking to this knowledge base the questions:

? :- POSSESSES(person$_2$, advertisement)
? :- LIKES-MORE-THAN(person$_2$, politician, advertisement)

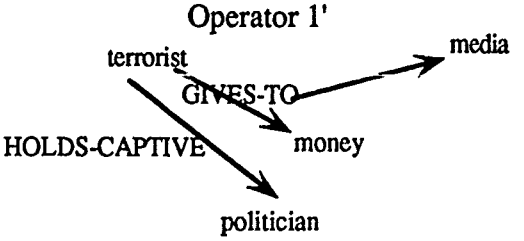
leads to a failure. It follows that we will be unable to apply OP$_1$.
Conversely, asking to this knowledge base the questions:

? :- GIVES-TO(person$_2$,x, advertisement)

gives two possible answers by using either clause 7 or clause 8.

Answer$_1$: GIVES-TO(x, media, money).
Answer$_1$': GIVES-TO(x, media, exciting-news).

One will notice that Answer$_1$' itself can lead to deeper problems by using clause 9. This is not our point here, we just want to find possible instantiations for OP$_2$, being understood that the application of OP$_2$ can lead to new problems. Answer$_1$' is such a case, and clause 9 is here to show how the next problems can be solved, but is not relevant to our present problem of analogy. Since we found the above two above answers, and that in both cases the variable person$_2$ was instantiated by media,we can claim that we are able to apply to the terrorist case, with the substitutions [person$_1$ ← terrorist, person$_2$ ← media, person$_3$ ←politician, advertisement(abstract) ← advertisement(concrete)].
Since we have already been unable to answer the questions ?POSSESSES(person$_2$, advertisement) and ?LIKES-MORE-THAN(person$_2$,politician,advertisement) it would be useless to attempt to ask them again by instantiating person$_2$ by media. Our solution is then to delete from OP$_1$ the links we have been unable to prove, to replace person$_2$ by media, and to replace the links between the characters in by those found as an answer to the application of OP$_2$. I follows that, in our example, the scheme found by analogy will be: Apply either OP$_1$ or OP'$_1$, as shown by figure 12 below, and then apply OP$_2$ with the instantiations [person$_1$ ← terrorist, person$_2$ ← media, person$_3$ ←politician, advertisement(abstract) ← advertisement(concrete)].



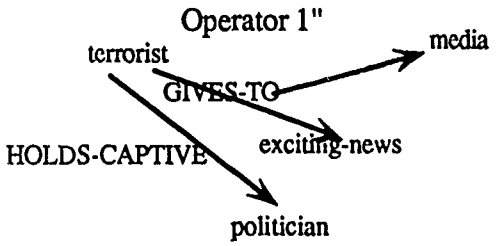


Figure 12. New operators obtained by deleting from OP_1 the failures, and adding the success coming from OP_2.

8 Conclusion

We have been proposing to use $\alpha \circ \beta \circ \alpha^{-1}$ as a combination of the causality β and of the similarity α in order to compute the causality β' which, in turn, allows to compute the missing part of the target. In simple cases, once the knowledge is properly represented, this computation is straightforward. On the contrary, when dealing with analogous solutions to similar problems, it might well happen that the computation of $\alpha \circ \beta \circ \alpha^{-1}$ fails, as exemplified in section 6. In such a case, we have to recover from the failure. The solution we suggest is to use the recently established techniques of abductive recovery (or, alternately, of inversion of resolution) in order to recover and find another plan able to solve the target problem.
Abductive recovery is not a process easy to implement, even though its basic mechanism, inversion of resolution has already been implemented twice (Muggleton and Buntine, 1988; Rouveirol and Puget, 1989). The implementation of the recovery process described in (Duval and Kodratoff, 1989) is presently under way, and will be used in order to recover form failures to perform an analogy.
In conclusion, we do believe that very deep analogies similar to the ones performed by humans are very far from the present state-of-the-art of artificial intelligence. On the contrary, simple and robust analogies performed by computing $\alpha \circ \beta \circ \alpha^{-1}$ are easy to implement, and, when the computation succeeds, they should become very soon a standard mechanism for performing simple changes allowing an adaptive behavior to vision and robotic systems.

References

Carbonell, J.G. Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition, in R.S. Michalski, J. G. Carbonell, T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach, Volume II*, Morgan Kaufmann 1986, pp. 371-392.

Carbonell, J.G. Learning by Analogy: Formulating and Generalizing Plans from Past Experience, in R.S. Michalski, J. G. Carbonell, T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Morgan Kaufmann 1983, pp. 137-159.

Chouraqui, E. Construction of a Model for Reasoning by Analogy, in *Progress in Artificial Intelligence*, L. Steels (Ed.), Halsted Press, New York, pp. 169-183, 1985.

Cox, P.T., Pietrzykowski, T. Causes for Events: Their Computation and Applications, *Proceedings of the Eighth International Conference on Automated Deduction*, Oxford, 1986, Lecture Notes in Computer Science n° 230, Springer Verlag, Berlin, pp. 608-621.

DeJong G.F. & Mooney R.J. Explanation-Based Learning: An Alternative View, *Machine Learning 1*, 2, pp.145-176, 1986.

Duval, B., Kodratoff, Y. A Tool for the Management of Incomplete Theories: Reasoning about explanations, in *Machine Learning, Meta-Reasoning and Logics*, P. Brazdil and K. Konolige (Eds), Kluwer Academic Press pp. 135-158, 1989.

Falkenhainer, B., Forbus, K. D., Gentner, D. "The Structure-Mapping Engine, Report N° UIUCDCS-R-86-1275, DCS, Univ. of Illinois at Urbana-Champaign, May 1986. See also Proc. AAAI-86.

Gentner, D., Analogical Inference and Analogical Access, in *Analogica*, Prieditis A. (Ed), Pitman, London, 1988, pp. 63-88.

Gentner, D., Structure-Mapping: A theoretical Framework for Analogy, *Cognitive Science 7*, pp. 155-170, 1983.

Kedar-Cabelli, S., Toward a Computational Model of Purpose-directed Analogy, in *Analogica*, Prieditis A. (Ed), Pitman, London, 1988, pp. 89-107.

Kodratoff, Y., *Introduction to Machine Learning*, Pitman, London, 1988.

Kodratoff, Y. Combining Similarity and Causality in Creative Analogy, Research Report 537, LRI, Jan. 1990.

Muggleton, S., Buntine, R. Machine invention of first order predicates by inverting resolution, *Proceedings of 5th International Machine Learning Workshop*, pp 339-352, Morgan Kaufmann, 1988.

Rouveirol, C., Puget J.F. A simple solution for Inverting Resolution, *Proceedings of the fourth European Working Session on Learning*, pp 201-211, Pitman, 1989.

Rouveirol, C., Puget J. F. A Simple Solution for Inverting Resolution, in *Proc. 4th EWSL*, Morik K. (Ed), Pitman, London 1989, pp. 201-210.

Russel, S. J., *The Use of Knowledge in Analogy and Induction*, Pitamn, London, 1989.

Winston, P. H. Learning new Principles from Precedents and Exercises, *Artificial Intelligence* 19, 1982, pp. 321-350.

Issues in the Design of Operator Composition Systems

Steven Minton
Sterling Federal Systems
AI Research Branch, Mail Stop: 244-17
NASA Ames Research Center
Moffett Field, CA 94035 U.S.A.

Abstract

Many learning problem solvers operate by composing operator sequences, so that a learned sequence can be applied as a unit during subsequent problem solving. In this paper we will describe an abstract model of the operator composition and problem solving processes, and use the model to analyze several design issues that affect the utility of the learning method. We will focus primarily on design issues that arose during the implementation of the PRODIGY system[16; 18] and two of its predecessors[17; 15]. The purpose of this paper is to consider these issues from the common perspective offered by our model, and to summarize the relevant research in the field.

1 Introduction

Many learning problem olvers employ the same general approach to learning from experience. They learn by composing rule sequences[2; 11], so that the sequences can be employed as single units during subsequent problem solving. Depending on the particular system, the composed sequence may be referred to as a macro-operator[5], a chunk[10], a heuristic[19], a search control rule[16], etc. Most of the literature in the field is concerned with how these composed sequences are learned. However, there are also important design issues that arise when one considers how the composed sequences should be stored and used during subsequent problem solving. Most of these issues have received little or no attention, though in fact they do influence the utility of the learning method and the circumstances under which the method is appropriate.

In this paper we suggest that operator composition has three primary effects on problem solving. Within the context of this model, we discuss how a variety of alternative methods for using composed sequences affect the problem-solving process. We focus primarily on design issues that arose while implementing three systems: PRODIGY[16; 18], MORRIS[17] and the CBG learning game-player[15]. The purpose of this paper is to consider these issues from the common perspective offered by our model, and to summarize the relevant research in the field, rather than to investigate any single approach or issue in depth.

2 Assumptions and Terminology

We can categorize problem-solving systems according to various criteria. The first is the domain specification language. Some systems use inference rules as their basic unit, as do theorem provers, in which case the learned rules can be stated as lemmas. Others use operators, as in STRIPS[5] or PRODIGY. For the purposes of this paper, we will assume only that the rules or operators have explicit preconditions and postconditions, and that they are composable. We will generically refer to these basic building blocks as *primitive operators*, and the structures composed from them as *macros*. We will restrict our attention to macros that represent simple operator sequences, and will not consider disjunctive or iterative macros[22]. The preconditions and postconditions of a learned macro are assumed to be exactly the the weakest preconditions and postconditions of the composed sequence of operators[16].[1]

For illustrative purposes, the primitive operators we will use in our examples will be simple STRIPS operators with conjunctive preconditions with variables, as these are very commonly employed. (Some systems, such as PRODIGY, use more expressive precondition languages and where this is relevant we will note this explicitly.) Our examples will be taken from robot problem-solving domains, similar to the STRIPS domain, that involve a single robot moving from room to room and accomplishing simple tasks. Since the operators and macros have variables, they actually represent operator schemas, and we will assume the variables must be bound before the operator can be applied. We will refer to an instance of an operator with all of its variables bound to constants as an *instantiation* of

[1] Often the primitive operators in a sequence can be composed in different ways, depending on how the preconditions and postconditions of the operators unify with each other. Thus several different macros may actually be composed from a given operator sequence.

that operator. In our figures, variables will be indicated in italics, and constants in uppercase.

A second criterion for categorizing problem solvers is the search method. Most systems use some variation of either forward search (as in production systems) or backward search (as in means-end analysis), with either a depth-first, breadth-first or best-first ordering strategy. In general, the issues we will discuss arise regardless of the search method, although there are some peculiarities that are method-specific, in which case these will be explicitly mentioned. We assume only that the problem solver operates by successively expanding the nodes of a search tree, so that at each node of the tree some set of primitive operators and macros is considered. Each edge in the tree thus corresponds to some operator or macro. Each node in the tree is a state in the search space.

3 The Effects of Macro Learning

The purpose of macro learning is generally to speed up problem solving. For example, in [16], the utility of a learned macro is measured in terms of its expected effect on problem-solving performance, as given by the following:

$$Utility = (AvrSavings \times ApplicFreq) - AvrMatchCost$$

AvrMatchCost is the expected time cost of determining whether the macro is applicable, *ApplicFreq* is the probability that the macro will be applicable when it is tested, and *AvrSavings* is the average time difference in problem-solving performance one can expect if the macro is applied, as compared to that if it is not applied. Note that the average savings can be negative, since after applying a macro the system might actually be farther from a solution than before. In general, the savings, match cost, and application frequency may depend on complex factors, such as the number of other learned macros in the system and the matching method. Thus the formula above gives the utility of an *individual* macro, but offers little insight into how well specific strategies for learning and using macros will perform. For this, one must consider the overall effect of macro learning on the search space. In this section, we suggest that macro learning can be viewed as having three primary effects.

First, macro learning changes the order in which the search space is traversed. Typically, operator sequences that have previously succeeded (i.e., those encoded as macros) are tried before other sequences. We refer to this as the *reordering effect* (or "experiential bias" [17]).[2] Figure 1 illustrates the reordering effect by showing the search tree for a simple depth-first, forward-chaining problem solver before and after learning. The search space contains five primitive

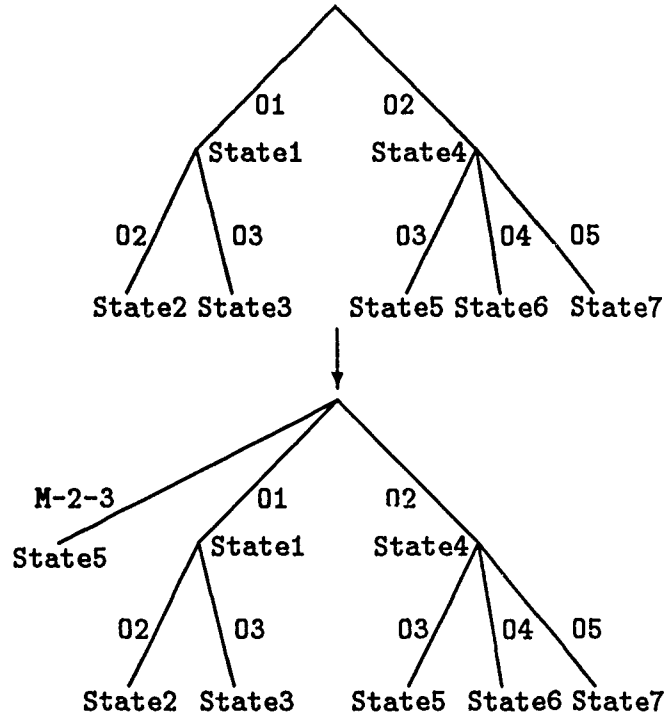


Figure 1: Illustration of the reordering effect for a very simple search space

operators, O1, O2, O3, O4 and O5, and one learned macro, M-2-3, created from the sequence O2, O3. (For simplicity, the figure only shows the top levels of the search tree.) During problem solving the system always tries applying the macro before any of the original primitive operators. In the figure, notice that state 5 is the fifth node visited before learning, but it is visited first after learning.

Another effect is a change in *path cost*: the cost of reaching a state via a macro may be more or less than the cost of testing and applying the corresponding sequence of primitive operators. Typically, the cost of using a macro is less than the cost of using the corresponding sequence of primitive operators. One reason is that there are often fewer preconditions and/or postconditions in the macro than in the corresponding sequence of primitive operators. For, example, consider a macro composed from the operator sequence GO-TO-OBJ, PICK-UP-OBJ. Most of the preconditions of the primitive operators are also preconditions of the macro. However, the precondition (NEXT-TO ROBOT *object*) of PICK-UP, for example, is not a precondition of the macro because it is added by the GO-TO-OBJ operator.

A third effect is that of *increased redundancy*, which tends to degrade performance. There are two sources of increased redundancy. First, after learning a set of macros, there may be many paths that lead to identical search states, since the the same sequence of primitive operators can be explored using various combinations of macros and primitive operators. Fig-

[2] In some systems, search is conducted exclusively with macros after a certain point, as in Korf's system[9], where the macros are guaranteed to find a solution. In any event, since we could theoretically search with the original system if the macros fail to find a solution, we can consider this as an extreme case of reordering.

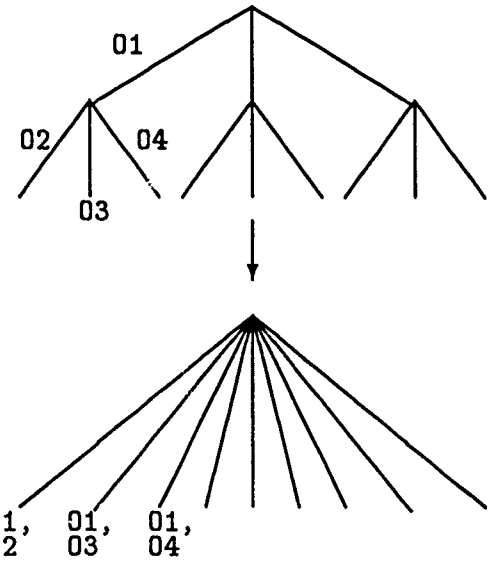


Figure 2: Complete conversion of search space into macros illustrates potential increase in redundancy

ure 1 illustrates this source of redundancy, which we refer to as *search state* redundancy. If macro M-2-3 does not lead to a solution, the corresponding sequence of primitive operators, O2, O3, will still be explored later in the search, and thus State5 (and all of its successors) will be visited twice. Second, the learned macros may have duplicate initial subsequences. For example, a macro composed of primitive operators O2 and O3 will share many of the same preconditions as a macro composed of primitive operators O2 and O4. Thus, the preconditions of a primitive operator, such as O2, may be tested again and again as each of the macros containing that operator are tested. We refer to this as *path* redundancy. Figure 2 shows the transformation produced by macro formation when all operator sequences are converted into macros, illustrating the potential increase in path redundancy. According to our model, increasing redundancy is one reason why a macro-learning system may perform *worse* than a non-learning system, as has been observed in a variety of empirical studies (e.g., [17; 13; 20; 14]).

These three factors often have conflicting effects on the performance of a macro system. For example, our experience with the MORRIS system [17], a STRIPS-like macro-learning system, indicated that the reordering effect had the highest potential for improving performance, but that increased redundancy could easily counter that if too many macros were learned. Path cost effects were much less significant than the other two effects.

We also found that in designing macro-learning systems, including PRODIGY, MORRIS and the CBG learning game-playing system, many design decisions had important ramifications with regard to the interaction of these effects and the resulting performance of the system. In the following sections, we will examine each of these effects in more detail, and how the design of a system influences the tradeoffs.

4 The Reordering Effect

To some extent, all macro-learning systems employ learned macros in preference to the original operators. Otherwise, it would be pointless to learn macros. Exactly what it means to "employ" a macro varies considerably, however. In this section we consider various schemes for using macros and their effect on the search process.

4.1 Intermediate States

The first reordering issue we will consider is whether the macro is employed as an indivisible atomic unit, or intermediate states are explored. Consider a simple forward search system, where macro M-1-2-3 is composed of individual operators O1, O2 and O3. As shown in figure 3, if the system discovers that the preconditions of macro M-1-2-3 are satisfied in the current state, then it can apply that macro, and arrive in state 4. However, if the goal can actually be achieved by just applying operators O1 and O2, then the system may have "jumped over" the solution, since the goal may not be true in state 4. (For a concrete example, consider a macro for moving a robot from one room into the next, and a problem where the goal is simply to have the robot be at the doorway.) An alternative option is, instead of treating the macro as a single atomic operator, to treat the macro as a sequence of primitive operators, so that intermediate states 2, 3 and 4 are visited in succession when the macro is applied. A similar option is available to backward chaining systems. When considering a macro that achieves some goal, a system can successively backchain on each operator in the sequence, rather than treating the macro as an indivisible operator.

This strategy may seem a bit odd at first, but in fact, this is essentially the way the STRIPS macro system operated. Given a macro for solving a particular goal, STRIPS would scan the macro to find the shortest subsequence whose preconditions were applicable, and thus avoid "jumping over" solutions. More specifically, given a macro representing operator sequence $O_1, O_2..O_n$, where O_n achieves the current goal, STRIPS would first check whether the preconditions of O_n matched, and if not, check the preconditions of O_{n-1}, O_n, and then O_{n-2}, O_{n-1}, O_n, and so on (using an efficient "planex scan" of the macro's triangle table) until the entire macro had been checked.

More sophisticated methods can be used to eliminate unnecessary operators in a macro. Consider a macro from the STRIPS robot-world in which the robot moves from one room into the next by going to the door, opening it, and moving through the door way. Given a problem where the door is already open, a simplistic STRIPS-like macro system will backchain on the preconditions of the macro, and have the robot first close the door, so that the macro can then be

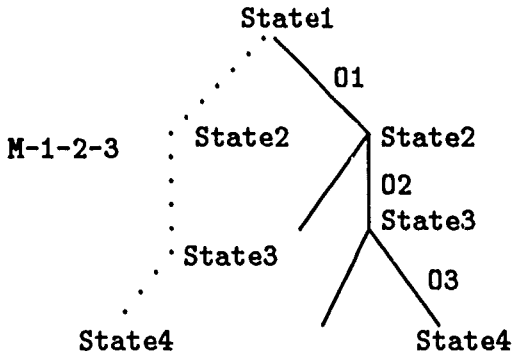


Figure 3: Jumping over states 2 and 3 with macro M-1-2-3

applied. A more sophisticated system that considers each operator in turn can eliminate the OPEN-DOOR operator from the plan since the door is already open.

Unfortunately there are generally two drawbacks to checking intermediate states. First, there is little point in compiling the preconditions and/or postconditions of the macro, since the individual operators must each be tested and applied in order to generate the intermediate states. In other words, macros lose their ability to decrease path cost. Second, redundancy is increased since the intermediate states may be visited multiple times. They will be visited when any macros with the same intermediate states are considered, as well as when the original primitive operators are explored.

4.2 The Single Operator Approach

Another common approach for employing macros also avoids the problem of jumping over intermediate states. This strategy is used by the PRODIGY system when it learns by observing successful operator sequences[16; 18]. PRODIGY'S explanation-based method[3] for learning from success is similar to macro-operator learning. (PRODIGY can also learn from problem-solving failures and goal interactions, however, we will focus only on learning from success, as it is most similar to traditional macro-operator learning.) To learn from a successful operator sequence, the system analyses the operator sequence in order to identify the sequence's preconditions. The resulting rule, called a search control rule, is very similar to a traditional macro-operator, but it is used differently. If a control rule is learned from an operator sequence, then it indicates a preference for the last operator in the sequence. (The last operator is chosen because PRODIGY employs means-ends analysis, a form of backward chaining). For example, consider a search control rule learned by the success of operator sequence O1, O2, O3 in solving a goal G. Although the preconditions of the control rule are the preconditions of the entire sequence, the control rule will recommend only that O3 be chosen to solve goal G. The obvious disadvantage of this strategy is that it can be expensive. After O3 is chosen, O2 and O1 still remain to be selected after the system backchains on O3. Thus multiple control rules are needed to exactly duplicate the action of a traditional macro. However, the theory behind this strategy assumes that at most decision points, control rules are unnecessary – the correct choice will be made through the use of the system's default heuristics. Control rules are only needed in specific circumstances where the wrong operator would be chosen, and a costly mistake would be made.

One advantage of this "single operator" strategy is that it allows control information to be brought to bear at intermediate states. Consider a situation where two macros exist, MA and MB, and a problem that involves multiple goals. Assume the solution requires interleaving macros MA and MB, e.g., applying an initial subsequence of MA, then applying MB, and finally finishing MA. If the system could only apply MA as a unit, it would never even notice that MB was applicable at an intermediate state. To be more concrete, macro MA might be "driving home from work", and macro MB might be "going to the bank to get money". Given the two goals of getting home and having money, one would want to consider the option of stopping at the bank on the way home from work. Single operator learning enables the system to take shorter paths through the state space when they arise serendipitously because the problem solver still plans one operator at a time. In the traditional macro approach, the problem solver is locked into pre-established sequences of operators.

A second advantage of the single operator approach is that it enables multiple rules to be combined together into simpler rules. For example, consider a system that has learned two macros, one for going to the car and opening the door (assuming it is unlocked) and another for going to the car, unlocking it, and opening it. These macros can be combined into an efficient set of single-operator rules, one for each step of the plan, rather than having a separate macro for each possible operator sequence. Several variations of the single operator approach have been developed, such as LEX2's use of problem-solving heuristics[19], and PET's use of episodes [21] (loosely packaged heuristic rules for operator selection). A variety of methods have been suggested for combining multiple rules, such as simplification (in PRODIGY) and induction (in LEX2). In-depth comparisons of these approaches have yet to be carried out.

[3]PRODIGY's search control rules are learned via explanation-based learning. The rules can be used to select, reject, or indicate preferences for operators, goals or bindings. In this paper we restrict the discussion to learning from successful operator sequences, which produces operator preference rules only. Although operator preference rules are used differently than macros, the EBL method is essentially identical to macro learning, and indeed, PRODIGY's EBL method for learning from successful operator sequences has also been used to produce traditional macros rather than search control rules[16].

4.3 Backchaining on Macros

Another reordering issue (related to our discussion of intermediate states) arises solely in connection with backward chaining systems. This is the question of whether a system should backchain on the preconditions of macros in addition to backchaining on the preconditions of primitive operators. In other words, if there exists a macro that can achieve a current goal (or subgoal), but its preconditions are not currently satisfied, should a system backchain on the unmatched preconditions of the macro? A variety of answers are possible. One strategy, used in PRODIGY,[4] is not to backchain on macros at all; if the preconditions of the macro are not applicable, then the macro is not used. A second, less restrictive strategy, used in STRIPS, is to backchain on the unmatched preconditions of the macro. A third strategy, even less restrictive, is to backchain on the unmatched preconditions of the "best" tail subsequence of a macro. (Given a macro composed of operators $O_1, O_2 ... O_n$, a tail subsequence is any subsequence $O_k, O_{k+1} ... O_n$, where $k < n$.) "Best" might be defined as the sequence with the fewest unmatched preconditions, for instance.

Empirical evidence from a study by Mooney[20] indicates that the strategy of backchaining on macros may be counterproductive, in that it will decrease utility. Mooney's study compared a system that backchained on macros to an extremely strict system that applied a macro only if it could solve the entire problem immediately (from the initial state). The second system tended to perform better. Unfortunately, Mooney's study did not consider intermediate points in the design space (such as a system that could apply macros to solve subgoals, but which would not backchain on the macros) and so the issue of backchaining on macros still has not been completely addressed. Experiments (unpublished) with the MORRIS system indicated that backchaining on macros can, by itself, reduce utility, which is consistent with Mooney's results.

Using our search space model it is straightforward to explain why backchaining on macros may be counterproductive. Consider a node in the search tree, as shown in figure 4. There will be a set of relevant macros at that node. (In a backward-chaining system, a rule or macro is relevant at a node if one or more of its postconditions unifies with one or more goals at that node.) Macros formed from similar subsequences, such as M-1-2-3 and M-2-3-4 in the figure, will contain many identical preconditions, such as P3. Each precondition is a potential subgoal. For each subgoal, there will be subtree (in the search tree) that must be explored in order to achieve that subgoal (or determine that it is unachievable). Thus, for each identical subgoal, there will be identical subtrees. In the figure, precondition P3 is an unmatched precondition of both macros, and therefore becomes a subgoal for both macros. Thus, this situation is one version of the redundancy problem described in section 3, since the same work will be repeated in different paths in the search space.

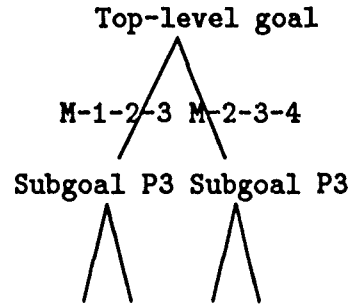


Figure 4: How backchaining on macros leads to redundancy

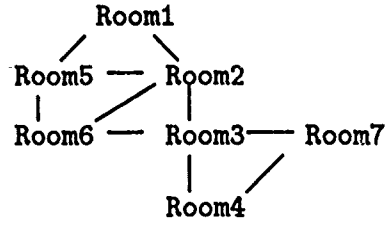
4.4 Interactions Between Search Strategy and Macro Usage

The last ordering issue we will consider in this section concerns the interaction of macro learning and depth-first search. For efficiency, many problem solvers use some variation of depth-first search. Unfortunately, macro learning can give a depth-first search a more breadth-first flavor, as the following example illustrates [16; 24]. Figure 5 shows a graph representing seven connected rooms in a house. Consider a simple operator for moving a robot from room to room: to move from *room-x* to *room-y*, the rooms must be directly connected (i.e., there must be an edge connecting them on the graph) and the robot must be in *room-x*. Given a problem where the robot must move to room 4 from room 1, a solution is to move from room 1, through rooms 2 and 3, and into room 4. A depth-first traversal of the search space will, in effect, explore the graph depth-first, which in this case is a very efficient way to find a path. The macro that is learned from this solution is shown in figure 5. It states that if there is a sequence of four connected rooms, then the robot can move from the first room to the last. Unfortunately, when using such a macro, most problem solvers will find *all* paths between rooms 1 and 4, which is considerably more expensive than finding a single path. This difficulty is due to the use of matching algorithms such as RETE [6], which attempt to find *all* matches to the preconditions of a macro. Thus, what was originally an efficient depth-first search is converted into a search where all paths up to length four will be explored (by the matcher).

One might argue that this problem can be solved simply by using a matcher that stops once a single successful instantiation of a macro is found. Unfortunately, the answer is not so simple. In a STRIPS-like macro system, for instance, a macro will rarely solve an entire problem, but will only solve one or more subgoals. Thus, the first instantiation found may solve a subgoal, but after applying the instantiated macro, it may not be possible to find a global solution. Alter-

[4] Prodigy can learn both traditional macros and search control rules, neither of which is backchained on.

Connections between rooms



Learned Macro

```
PRECONDITIONS:(AND (INROOM ROBOT w)
                   (CONNECTED w x)
                   (CONNECTED x y)
                   (CONNECTED y z))
POSTCONDITIONS: (ADD (INROOM ROBOT z))
                (DELETE (INROOM ROBOT w))
```

Figure 5: Room connections, and a macro for moving between rooms

native instantiations must then be explored. Consider a slightly more complex domain in which some of the rooms have wet floors. In this domain the move operator is augmented so that moving into a room with wet floors will make its wheels wet. Consider a problem where the robot must be moved from room 1 to room 4 so that it can accomplish some task in room 4 (e.g., moving a box), but now room 2 is wet. Unfortunately, after taking the first path found to room4 (though rooms 2 and 3) the problem solver may discover that the robot cannot accomplish its task because it lacks traction due to its wheels being wet. The problem solver must then backtrack and consider alternative paths to room 4, until it finds one that leads through dry rooms, so that a complete solution can be generated. Thus, an efficient macro problem solver should stop matching as soon as an appropriate instantiation is found, but be capable of resuming the matching process if that instantiation does not lead to a solution.

Finally, we note that this issue is even more problematic for problem solvers, such as MORRIS, that use heuristic evaluation to determine which operator or macro to select. Such problem solvers normally attempt to find all instantiations of relevant operators at each decision point so that they can be evaluated. In other words, the system must find all instantiations of all operators and macros, and then choose the best alternative. This exacerbates the matching problem, since one cannot use the method described above in which matching terminates after a single instantiation is found. In such systems, macro learning can seriously distort the depth-first flavor of the search, since the matching process cannot proceed in a depth-first manner.

5 Decreasing Path Cost

Once a macro has been learned, one can "compile" the macro by reformulating, rearranging or indexing its preconditions and/or postconditions in order to decrease the cost of using the macro. Compilation can reduce the cost of testing and applying a macro significantly when compared to the cost of successively testing and applying the corresponding sequence of primitive operators.

Much of the work in this area has concentrated on reducing the cost of matching the preconditions of macros, because precondition matching is typically extremely expensive. In fact, precondition matching is NP-complete, assuming the preconditions are each existentially quantified conjuncts [7; 16]. (Precondition matching is even more expensive if the precondition language allows arbitrary existential and universal quantification; in this case the task is P-space complete.) Many standard matching algorithms, including the matching algorithm used in PRODIGY[16] and the RETE matching algorithm [6] used by SOAR[10], run in time $O(s^p)$ in the worst case, where s is the number of conditions that are true in the state, and p is the number of preconditions in the macro.

The cost of matching is particularly important for macro systems, because macros can have many preconditions. If a macro is created from a primitive operator sequence $O_1, O_2 ... O_n$, then every precondition of every primitive operator in this sequence will also be a precondition of the resulting macro, except for preconditions that are guaranteed to be true due to the action of earlier operators in the sequence. (A precondition of O_k is guaranteed to be true if an earlier operator O_j, $j < k$, either tests or adds that same precondition, and no operator between O_j and O_k can delete that condition.) Thus the number of preconditions in a macro will normally be less than the sum of the preconditions of the primitive operators from which it is composed, but the difference is often relatively small. In general, a macro may have up to $(P-1) \times N$ preconditions, where N is the length of the operator sequence, and P bounds the number of preconditions in an operator. This assumes that each operator in the original sequence, except for the last, makes at least one precondition of a subsequent operator true.[5] Therefore methods for reducing the matching cost for macros can be quite important. A great deal of attention was focused on this in the PRODIGY

[5]This formula also assumes that there is a single goal that the macro is designed to make true, and that the macro's preconditions are formed solely from the preconditions of the primitive operators in the sequence. (In some systems, extra preconditions must be inserted in the macro in order to preserve correctness. For example, in STRIPS-like systems, a macro may contain additional preconditions such as (NOT-EQUAL a b), which specifies that variables a and b cannot have the same value, in order to guarantee that the postconditions of the macro operate properly.)

system, where the process of reducing match cost was referred to as "compression".

The most obvious technique for reducing the cost of testing a list of preconditions is to order the preconditions in an intelligent manner. As discussed in [16], if the preconditions of a macro are simply ordered so that they will be tested in the same order as in the corresponding conditions in the primitive rule sequence, then the cost of testing the preconditions of the macro will approximate the cost of searching in the original search space.[6] Many macro-learning systems, including MORRIS, PRODIGY and SOAR, employ heuristic techniques to generate a better order. Optimal precondition ordering is generally not guaranteed, as it requires knowledge about the domain which is typically unavailable; for example, it is necessary to know the probability that a condition will match, given that other conditions have, or have not, successfully matched. Although precondition ordering techniques are commonly used to improve matching cost, the problem has received very little attention in the AI literature as compared to the well-studied problem of finding good subgoal orderings for problem solving and inference (e.g., [4; 23]). However, the problem of optimizing precondition ordering for matching has received significant attention in the database literature, where it is considered a part of conjunctive query optimization[25].

In addition to ordering the preconditions of a macro, it is often possible to simplify the preconditions, thereby reducing their match cost. For example, table 1 shows the preconditions for a macro composed of operators GO-TO-OBJ, PICKUP-OBJ, GO-TO-DOOR, PUTDOWN-NEXT-TO, which is learned from a problem where the robot must position an object next to the door. (The operator definitions are taken from [16]). Next to each precondition is shown the operator from which it came. These preconditions can be simplified considerably, as shown in the table. Notice that the precondition (IS-OBJECT x), for instance, is unnecessary and can be removed, because (IS-CARRIABLE x) is also a precondition, and anything that is carriable is an object. Also notice that if (IN-ROOM ROBOT ry) is unnecessary given that (IN-ROOM ROBOT rx) is true, since the robot can only be in one room at a time.[7] As even this simple example illustrates, simplification and reordering are most useful when the operators in a macro constrain each other. In our example, the fact that the object must be carriable and in the same room as the robot may greatly constrain the choice of object. Thus applying this macro may be significantly more efficient than problem solving. For instance, consider a forward search problem solver. When applying GO-TO-OBJ the system must select an object to go to, and an arbitrary object in the room will be chosen; it is only when the system attempts to apply PICK-UP-OBJ that the CARRIABLE constraint is encountered, in which case backtracking will be necessary if the object cannot be carried. In contrast, after learning the macro, the preconditions can be ordered so that the choice of an object is immediately determined by the CARRIABLE constraint. (If the preconditions are not ordered appropriately then inefficiencies will result. For instance, if the preconditions are ordered in the same order that the problem solver encounters them, then the matcher can recreate the same binding mistake as the problem solver! This subject is discussed at length in [16].)

BEFORE:	
(IS-OBJECT x)	from GO-TO-OBJ
(IN-ROOM x rx)	from GO-TO-OBJ
(IN-ROOM ROBOT rx)	from GO-TO-OBJ
(ARMEMPTY)	from PICKUP-OBJ
(CARRIABLE x)	from PICKUP-OBJ
(IN-ROOM ROBOT ry)	from GO-TO-DOOR
(IS-DOOR dr)	from GO-TO-DOOR
(DR-TO-RM dr ry)	from GO-TO-DOOR
(IN-ROOM ROBOT rx)	from PUTDN-NXT-TO
(IS-OBJECT dr)	from PUTDN-NXT-TO
(IN-ROOM ry dr))	from PUTDN-NXT-TO

AFTER:
(IN-ROOM ROBOT rx)
(IN-ROOM x rx)
(CARRIABLE x)
(ARMEMPTY)
(DR-TO-RM dr rx))

Table 1: The preconditions of a macro, before and after simplification and reordering

There are various techniques for simplifying precondition expressions, depending on how much knowledge is available. If arbitrary inference rules are available to the simplifier, then producing the least expensive precondition expression requires theorem-proving [16]. Unfortunately, theorem-proving is undecidable. Less expensive simplification techniques include partial evaluation, and the application of heuristic transformations. All three of these techniques were applied by PRODIGY's compression module, although the theorem-prover was extremely restricted.

Other techniques for reformulating preconditions are also available. For example, Chase et al. [3] have described a technique that takes a precondition expression and drops conjuncts that are expensive to evaluate. The purpose is to find an approximation to the original expression that is less expensive to match.

[6] We note that the cost of matching a list of preconditions depends not only on the number of preconditions, but also on the number of bindings generated for the variables in the preconditions. However, as described in [16], the number of bindings generated when matching a macro's preconditions can be expected to be the same as the number of bindings generated when matching the corresponding operator sequence, assuming that the conditions are tested in the same order.

[7] If the robot could be in more than one room at a time, these preconditions would not be redundant, since rx and ry have different restrictions.

Current results indicate that their technique leads to only minor improvements in efficiency, but the approach appears promising. Techniques for indexing macros, which effectively reduce the average time to check whether a macro is applicable, are also an area of current interest (e.g., [1]). Indexing combines aspects of precondition reordering (in that the most important conditions are examined first) and structure sharing (in that many macros can share indices) to improve efficiency.

6 Eliminating Redundancy

A significant problem with using macro-operators is that they introduce redundancy. As described earlier, macros introduce redundancy in two ways. First, states may be searched repeatedly. Secondly, macros which represent common subpaths may have duplicate substructure.

One way to reduce the first type of redundancy is simply to record each search state so that it is not visited more than once. This approach was investigated by Markovitch and Scott[14], who pointed out some obvious problems. First, the bookkeeping costs are high, especially the space cost. Secondly, the problem is not completely eliminated, since the problem solver would still visit some states more than once (although it would recognize that such a state had already been visited, and discontinue search along that path). Nevertheless, Markovitch and Scott did implement this solution, and found that performance of their system was improved by a factor of two.

A method for reducing the second type of redundancy is to employ a scheme for sharing common substructure within macros. For example, if two macros share several preconditions, we can separate them into three structures, one intermediate structure with the shared preconditions, and two others each with the remaining conditions. This way the shared preconditions in the intermediate structure need only be matched once. Wogolus and Langley [26] have suggested such a scheme for creating intermediate concepts from an initial domain theory. Such techniques would presumably be even more useful once macros have been learned, due to the increase in redundancy.

A limited form of substructure sharing is automatically implemented within the RETE match network[6], upon which the SOAR system is built. If SOAR contains a chunk with preconditions A, B, C, D, E and another chunk with preconditions A, B, C, F, G, H, then the network will automatically create a shared structure for matching conditions A,B and C. Unfortunately, if the second chunk is H, A, B, C, D, the network will not create shared substructure, since it will only do so if it can find an identical *initial* subsequence.

More complex methods for sharing substructure are of course possible, but little research has been done. Methods for maximizing shared substructure may be quite complex. Moreover, unfortunately, such methods generally conflict with compilation schemes for reducing path cost (such as those discussed in the previous section). For example, maximizing substructure sharing may conflict with optimally ordering the conditions in a macro. Thus, if we have two macros with preconditions A, B, C, H, and preconditions A, B, C, J, it may be that, individually, the best ordering of their preconditions is A, H, C, B and A, B, J, C, respectively, which reduces the amount of sharing that is possible. Similarly, methods for simplifying the preconditions of macros may destroy possibilities for sharing.

Interestingly, sharing of preconditions via the creation of intermediate concepts tends to recreate the structure of the original search space. This can be seen in figure 2. Whereas complete macro formation will convert the space shown on the top of the figure into the space shown on the bottom the figure, the creation of intermediate concepts tends to work in exactly the opposite direction, i.e., it will convert the space shown on the bottom of the figure into the space shown on the top of the figure. For this reason, Wogulis and Langley describe their method for creating intermediate concepts as the inverse of explanation-based learning. However, when both techniques are used in conjunction, in a discriminating way, the advantages of both can be combined (as was found with the CBG gameplayer[15] where structure sharing was implemented by hand).

The most intensively investigated technique for reducing redundancy is to limit the number of macros that are used by the system [17; 16; 13; 27; 8]. (As this subject has received considerable attention elsewhere, we will only briefly touch upon it.) There are a variety of methods for selecting the most useful macros. Frequency of use, heuristic informativeness, and a variety of other utility metrics have all been investigated. Markovitch and Scott[12] introduce a taxonomy of methods for limiting macro use. Depending on when the "filtering" process takes place, they distinguish between selective experience, selective attention, selective acquisition, selective retention, and selective utilization. These methods have a common purpose – by restricting a system's consideration to the most useful macros, not only is redundancy limited, but the ordering bias is improved so that the most useful paths in the search space are explored first.

7 Conclusion

This paper has introduced a model that suggests that macro learning has three primary effects: the search space is reordered, the path cost of reaching particular states may change, and redundancy may increase. The model represents a first step towards a predictive theory of the utility of macro learning. Previous work has indicated that the utility of macro learning is high when a small set of macros can be learned that are sufficient for solving most problems. In this case, according to our model, the positive effect of reordering is maximized, while the negative effect of redundancy is minimized. In this paper, we have examined a variety of more detailed issues in the design of macro systems, and the complex tradeoffs that determine the utility of macro learning. Eventually, we hope that through

a combination of additional empirical studies, and refinements to the model, we will be able to validate the model and make quantitative predictions concerning the performance of macro-learning techniques.

8 Acknowledgments

Thanks to Pat Langley, Monte Zweben, Richard Keller, John Allen and Bernadette Kowalski for their helpful comments and suggestions.

References

[1] J.A. Allen and P. Langley. Using concept hierarchies to organize plan knowledge. In *Proceedings of the Sixth International Machine Learning Workshop*, 1989.

[2] J.R. Anderson. Knowledge compilation: The general learning mechanism. In *Machine Learning II*, Tioga Press, Palo Alto, CA, 1985.

[3] M.P. Chase, M. Zweben, R.L. Piazza, J.D. Burger, P.P. Maglio, and H. Hirsch. Approximating learned search control knowledge. In *Proceedings of the Sixth International Machine Learning Workshop*, 1989.

[4] J. Cheng and K.B. Irani. Ordering problem subgoals. In *IJCAI-89*, Detroit, MI, 1989.

[5] R. Fikes, P. Hart, and N. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4), 1972.

[6] C. Forgy. Rete: A fast algorithm for the many pattern/many object pattern matching problem. *Artificial Intelligence*, 19(1), 1982.

[7] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., 1979.

[8] G.A. Iba. A heuristic approach to the discovery of macro-operators. *Machine Learning*, 3(4), 1989.

[9] R.E. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26(1), 1985.

[10] J.E. Laird, P.S. Rosenbloom, and A. Newell. Chunking in soar: the anatomy of a general learning mechanism. *Machine Learning*, 1(1), 1986.

[11] C. Lewis. Composition of productions. In *Production System Models of Learning and Development*, MIT Press, Cambridge, MA, 1987.

[12] S. Markovitch and P.D. Scott. Information filters and their implementation in the syllog system. In *Proceedings of the Sixth International Machine Learning Workshop*, 1989.

[13] S. Markovitch and P.D. Scott. The role of forgetting in learning. In *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, MI, 1988.

[14] S. Markovitch and P.D. Scott. Utilization filtering: A method for reducing the inherent harmfulness of deductively learned knowledge. In *IJCAI-89*, Detroit, MI, 1989.

[15] S. Minton. Constraint-based generalization. In *AAAI-84*, Austin, TX, 1984.

[16] S. Minton. *Learning Search Control Knowledge: An Explanation-based Approach.* Kluwer Academic Publishers, Boston, Massachusetts, 1988. Also available as Carnegie-Mellon CS Tech. Report CMU-CS-88-133.

[17] S. Minton. Selectively generalizing plans for problem solving. In *IJCAI-85*, Los Angeles, CA, 1985.

[18] S. Minton, J.G. Carbonell, C.A. Knoblock, D.R. Kuokka, O. Etzioni, and Y. Gil. Explanation-based learning: a problem solving perspective. *Artificial Intelligence*, 40, 1989.

[19] T. Mitchell, P. Utgoff, and R. Banerji. Learning by experimentation: acquiring and refining problem-solving heuristics. In J. Carbonell, R. Michalski, and T. Mitchell, editors, *Machine Learning*, Tioga Publishing Co., 1983.

[20] R. Mooney. The effect of rule use on the utility of explanation-based learning. In *IJCAI-89*, Detroit, MI, 1989.

[21] B. Porter and D. Kibler. Experimental goal regression: A method for learning problem-solving heuristics. *Machine Learning*, 1(3), 1986.

[22] P. Shell and J. Carbonell. Towards a general framework for composing disjunctive and iterative macro-operators. In *IJCAI-89*, Detroit, MI, 1989.

[23] D.E. Smith and M.R. Genesereth. Ordering conjunctive queries. *Artificial Intelligence*, 26, 1989.

[24] M. Tambe and P. Rosenbloom. Eliminating expensive chunks by restricting expressiveness. In *IJCAI-89*, Detroit, MI, 1989.

[25] J.D. Ullman. *Principles of Database and Knowledge-base Systems.* Computer Science Press, 1989.

[26] J. Wogolus and P. Langley. Improving efficiency by learning intermediate concepts. In *IJCAI-89*, Detroit, MI, 1989.

[27] S. Yamada and S. Tsuji. Selective learning of macro-operators with perfect causality. In *IJCAI-89*, Detroit, MI, 1989.

Incremental Learning of Explanation Patterns and their Indices

Ashwin Ram

Georgia Institute of Technology
School of Information and Computer Science
Atlanta, Georgia 30332-0280
(404) 853-9372
ashwin@gatech.edu

Abstract

This paper describes how a reasoner can improve its understanding of an incompletely understood domain through the application of what it already knows to novel problems in that domain. Recent work in AI has dealt with the issue of using past explanations stored in the reasoner's memory to understand novel situations. However, this process assumes that past explanations are well understood and provide good "lessons" to be used for future situations. This assumption is usually false when one is learning about a novel domain, since situations encountered previously in this domain might not have been understood completely. Instead, it is reasonable to assume that the reasoner would have gaps in its knowledge base. By reasoning about a new situation, the reasoner should be able to fill in these gaps as new information came in, reorganize its explanations in memory, and gradually evolve a better understanding of its domain.

We present a story understanding program that retrieves past explanations from situations already in memory, and uses them to build explanations to understand novel stories about terrorism. In doing so, the system refines its understanding of the domain by filling in gaps in these explanations, by elaborating the explanations, or by learning new indices for the explanations. This is a type of *incremental learning* since the system improves its explanatory knowledge of the domain in an incremental fashion rather than by learning new XPs as a whole.

1 Case-based learning

Case-based reasoning and learning programs deal with the issue of using past experiences or *cases* to understand, plan for, or learn from novel situations [Kolodner, 1988; Hammond, 1989]. This happens according to the following process: (a) Use problem description to get reminded of old case. (b) Retrieve the results (lessons, explanations, plans) of processing the old case and give them to the understander, planner or problem-solver. (c) Adapt the results from the old case to the specifics of the new situation. (d) Apply the adapted results to the new situation.

The intent behind case-based reasoning is to avoid the effort involved in re-deriving these lessons, explanations or plans by simply reusing the results from previous cases. However, this process assumes that past cases are well understood and provide good "lessons" to be used for future situations, since it is these very cases that determine the performance of the system in new situations. This assumption is usually false when one is learning about a novel domain, since cases encountered previously in this domain might not have been understood completely. Instead, it would be reasonable to assume that the reasoner would have *gaps* in the knowledge represented by these cases.

Even if past cases are not well understood, they can still be used to guide processing in new situations. However, in addition to using the past case to understand the *new* situation, a reasoner can also learn more about the *old* case itself, and thus improve its understanding of the domain. This is an important problem that has not been addressed in case-based reasoning research, and one that is suited to a machine learning approach in which learning occurs incrementally as these gaps are filled in through experience.

This paper describes a case-based story understanding system that retrieves past explanations from situations already in memory, and uses them to build explanations to understand novel situations encountered in newspaper stories about terrorism. The system learns in an incremental manner, by filling in the gaps in the retrieved explanation that is being used as a precedent in understanding the new situation. What is done with the newly learned information depends on the kind of "knowledge gap" the system is trying to fill. The new piece of knowledge could result in a new explanation in memory; it could be used to fill in a gap in an existing explanation; it could be used to elaborate an existing explanation if that explanation was not detailed enough to deal with the new situation; or it could be used to reorganize or re-index knowledge in memory to allow the reasoner to use what it already knows in novel situations to which that piece of knowledge had

not been applied before. Each type of learning leaves the system a little closer to a complete understanding of its domain. Each type of learning could also result in a new set of gaps as the system realizes what else it needs to learn about, which in turn drives the system towards further learning.

Much of real-world learning is an incremental process of this type. A reasoner learns by modifying what it already knows using little pieces of new information that it comes across during its experiences. This paper presents a theory of incremental learning for case-based story understanding.

2 Explanation patterns

Before we can discuss the learning process, we must describe what needs to be learned. This depends on the purpose to which the learned knowledge will be put. Consider the problem of building motivational explanations for the purpose of understanding stories. An understander could construct such explanations by using rules connecting typical goals and plans of people (e.g., [Wilensky, 1978]). However, this would be very inefficient in complicated situations, where motivational causal chains could be several steps long. To get around this problem, a case-based understander uses pre-stored explanations for stereotypical situations. These explanations represent standard patterns that are observed in these situations, and hence are called *explanation patterns* [Schank, 1986]. When the understander sees a situation for which it has a canned explanation pattern (XP), it tries to apply the XP to avoid detailed analysis of the situation from scratch. Thus an XP is like an abstract case; it represents a generalization based on the understander's experiences that can be used as a paradigmatic case for similar situations in the future.

For example, a "blackmail" situation may be represented by the following XP (`xp-blackmail`):[1]

(1) The blackmailee has a goal G1.

(2) The blackmailer has a goal G2, and the blackmailee does not have the goal G2 (since otherwise he or she would satisfy the goal without needing to be threatened).

(3) The blackmailee has a goal G3, which he or she values above goal G1.

(4) The blackmailer threatens to violate G3 unless the blackmailee performs an action A that satisfies G2, even though the action would have a negative effect of violating G1.

3 Learning explanation patterns

How are stereotypical XPs formed in memory? The work in explanation-based learning focusses on the problem of learning through the generalization of causal structures underlying novel situations [DeJong and Mooney, 1986; Mitchell *et al.*, 1986]. However, it is difficult to determine the correct level of generalization. Furthermore, many stories do not provide enough information to prove that the explanation is correct. The understander must often content itself with two or more competing hypotheses, or otherwise jump to a conclusion. This means that in a real world situation, an explanation-based learning system may still need to deal with the problem of incomplete or incorrect domain knowledge.

Thus the system's memory of past experiences will not always contain "correct" cases or "correct" explanations, but rather one or more hypotheses about what the correct explanation might have been.[2] These hypotheses often have questions attached to them, representing what is still not understood or verified about those hypotheses. As the understander reads new stories, it is reminded of past cases, and of old explanations that it has tried. In attempting to apply these explanations to the new situation, its understanding of the old case gradually gets refined. New indices are learned as the understander learns more about the range of applicability of the XP. The XP is re-indexed in memory and is more likely to be recalled only in relevant situations.

Thus XP learning is an incremental process of theory formation, involving both case-based reasoning and explanation-based learning processes.

4 The AQUA program

AQUA is a story understanding program which learns about terrorism by reading newspaper stories about terrorist incidents in the Middle East [Ram, 1987; Schank and Ram, 1988; Ram, 1989]. AQUA reads stories about suicide bombing and attempts to understand them by constructing causal and motivational explanations for the events in the stories.

AQUA's case memory is based on XPs that have been used to explain past situations. AQUA improves its explanatory knowledge of the domain through a process of re-indexing and incremental modification of these XPs. For example, suppose AQUA has just read the following suicide bombing story (New York Times, April 14, 1985):

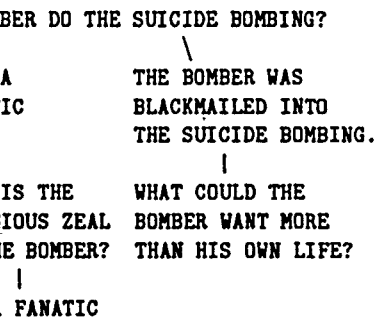
> **Boy Says Lebanese Recruited Him as Car Bomber.**
>
> JERUSALEM, April 13 — A 16-year-old Lebanese was captured by Israeli troops hours before he was supposed to get into an explosive-laden car and go on a suicide bombing mission to blow up the Israeli Army headquarters in Lebanon. ...
>
> What seems most striking about [Mohammed] Burro's account is that although he

[1] Details of XP representations may be found in [Ram, 1989].

[2] Actually, a single story or episode can provide more than one "case," each case being a particular interpretation or dealing with a particular aspect of the story. For an explanation program, each anomaly in a story, along with the corresponding set of explanatory hypotheses, can be used as a case.

> is a Shiite Moslem, he comes from a secular family background. He spent his free time not in prayer, he said, but riding his motorcycle and playing pinball. According to his account, he was not a fanatic who wanted to kill himself in the cause of Islam or anti-Zionism, but was recruited for the suicide mission through another means: blackmail.

After reading this story, AQUA builds the following *hypothesis tree* in memory, representing an anomaly (*Why would the bomber perform an action that resulted in his own death?*), alternative hypotheses constructed by applying known XPs to the anomalous situation (*religious fanatic* and *blackmail*), questions that would verify these hypotheses, and answers to these questions, if any:[3]



The final explanation built for this story involves a novel application of a stereotypical XP, `xp-blackmail`. Even though the system already knows about blackmail, it learns a new variant of this XP (`xp-blackmail-suicide-bombing`), based on the particular manner in which `xp-blackmail` was adapted to fit the story. AQUA also learns indices to the new XP. Both kinds of learning are important in a case-based reasoning system. Let us start with the latter.

5 Learning indices for explanation patterns

Regardless of whether a new XP is learned from scratch or by applying an existing XP to a new situation, the XP needs to be indexed in memory appropriately so that it can be used in future situations in which it is likely to be useful. Ideally, an XP should be indexed in memory such that it is retrieved only in those situations in which it is applicable. But this is impossible in practice. For example, consider the applicability conditions for "blackmail." In general, blackmail is a possibility whenever "someone does something he doesn't want to do because not doing it results in something worse for him." But trying to show this in general is very hard. Thus, in addition to general applicability conditions, an understander must learn specific, sometimes superficial, features that suggest possibly relevant XPs even though they may not completely determine the applicability of the XP to the situation. For example, a classic blackmail situation is one where a rich businessman who is cheating on his wife is blackmailed for money using the threat of exposure. If one read about a rich businessman who suddenly began to withdraw large sums of money from his bank account, one would expect to think of the possibility of blackmail. However, one does not normally think of blackmail when one reads a story about suicide bombing, although theoretically it is a possible explanation.

The point is that XPs are associated with stereotypical situations and people in memory. An understander needs to learn the stereotypical categories that serve as useful indices for motivational explanations. This is a type of inductive category formation [Dietrich and Michalski, 1981]; however, the generalization process is constrained so that the features selected for generalization are those that are causally relevant to the explanations being indexed [Flann and Dietterich, 1989].

AQUA indexes motivational XPs in memory using typical contexts in which the XPs might be encountered (*situation indices*), as well as character stereotypes representing typical categories of people to whom the XPs might be applicable (*stereotype indices*) [Ram, 1989]. In the above example, AQUA learns a new context for blackmail (suicide bombing), as well as a new character stereotype representing the type of person who one might expect to see involved in a "blackmailed into suicide bombing" explanation. Let us discuss how AQUA learns these indices.

5.1 Learning situation indices

AQUA learns new contexts (e.g., "suicide bombing") for stereotypical XPs (e.g., "blackmail") which are then used as situation indices for these XPs in the future. The main issue here is how far the context should be generalized before it is used as an index. In the above example, should the new situation index for blackmail be `suicide-bombing`, `suicide`, `bombing`, `destroy`, or indeed any MOP (action) with a negative side effect for the actor? The issue here isn't one of correctness but of utility. As discussed earlier, `xp-blackmail` is a possibility whenever the actor does something he would ordinarily not do because of a negative side effect. However, XP theory tries to replace generalized reasoning of this form with specific reasoning about stereotypical situations. The latter is more efficient even though it is less general.

After reading the above story, for example, one would expect to think of blackmail when one reads another story about a suicide bombing attack. However, one would probably not think of blackmail on reading any story about `suicide`, say, a teenager killing himself after failing his high school examinations, even though theoretically it is a possible explanation. Furthermore, it would not be useful to index the new XP under `bombing` in general (as opposed to `suicide-bombing` in particular), since the particular goal violation of the `p-life` goal is central to this explanation.

Thus in the above example, AQUA uses `suicide-`

[3] The understanding process by which AQUA builds this hypothesis tree is irrelevant for the purposes of this paper. Details may be found in [Ram, 1989].

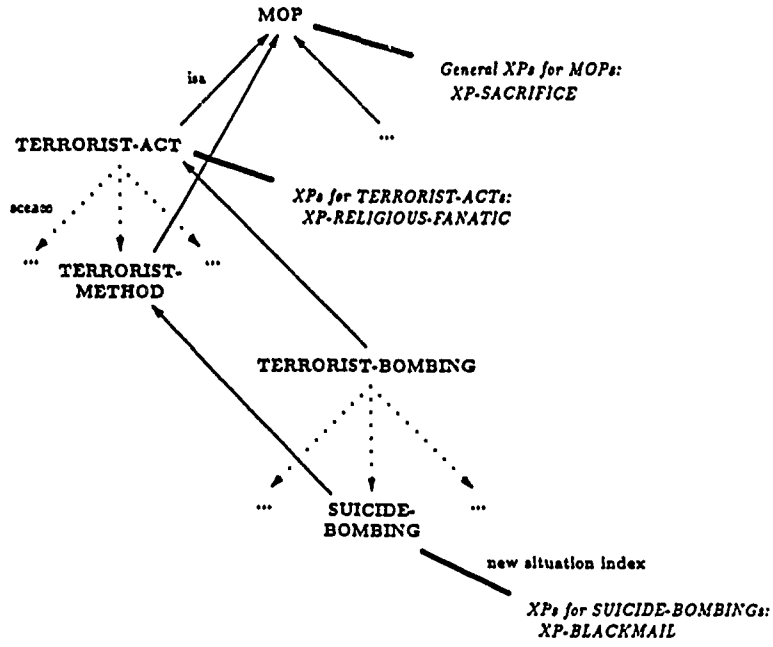


Figure 1: Learning situation indices for XPs. Upward lines represent **isa** links, and downward dotted lines represent **scenes** of MOPs. Heavy lines represent situation indices, which point from MOPs to XPs. Here, AQUA has just built a situation index from **suicide-bombing** to a copy of **xp-blackmail**.

bombing as the situation index for the new variant of **xp-blackmail** (figure 1). After reading several stories about blackmail, AQUA would know about different stereotypical situations in which to use the blackmail explanation, rather than a generalized logical description of every situation in which blackmail is a possible explanation. In other words, AQUA would have indexed a copy of **xp-blackmail** under all the MOPs for which it has seen **xp-blackmail** used as an explanation. Whenever these MOPs are encountered, AQUA would retrieve the new blackmail XP (if the other indices are also present).[4] The reason that a copy of the original XP is used is that the XP, once copied, will need to be modified for that particular situation, as discussed below.

5.2 Learning stereotype indices

The main constraint on a theory of stereotype learning is that the kinds of stereotypes learned must be useful in retrieving explanations. In other words, they must provide the kinds of discrimination that are needed for indexing XPs in memory. Since volitional explanations are concerned with goals, goal orderings, plans and beliefs of characters, the learning algorithm must produce typical collections of goals, goal-orderings, plans and beliefs, along with predictive features for these elements. Such a collection is called a *character stereotype*.

Character stereotypes serve as motivational categories of characters and are an important index for XPs in memory. In the above example, AQUA learns a new stereotype (**stereotype.79**) representing a typical Lebanese teenager who might be blackmailed into suicide bombing, which is used to index the blackmail XP. The stereotype is built from the novel blackmail explanation by generalizing the features of the character involved in that explanation:

```
Answering question: WHY DID THE BOY DO THE SUICIDE
                    BOMBING?
              with: THE BOY WAS BLACKMAILED INTO
                    DOING THE SUICIDE BOMBING.

Novel explanation for A SUICIDE BOMBING!

Building new stereotype STEREOTYPE.79:

    Typical goals:
        P-LIFE (in)
        A-DESTROY (OBJECT) (out)
        AVOIDANCE-GOAL (STATE) (question)
    Typical goal-orderings:
        AVOIDANCE-GOAL (STATE) over P-LIFE (question)
```

[4] AQUA can still understand other blackmail situations that it has not learned about as yet, as it did the story in the above example. Thus not having a situation index for an XP does not necessarily mean that the XP cannot be applied to the situation, but rather that this XP is not one that would ordinarily come to mind in that situation.

```
    Typical beliefs:
        RELIGIOUS-ZEAL = NOT A FANATIC (in)
    Typical features:
        AGE = TEENAGE AGE (hypothesized)
        RELIGION = SHIITE MOSLEM (hypothesized)
        GENDER = MALE (hypothesized)
        NATIONALITY = LEBANESE (hypothesized)

Indexing XP-BLACKMAIL-SUICIDE-BOMBING
    Stereotype index = STEREOTYPE.79
    Situation index  = SUICIDE-BOMBING
```

The label in (out) marks features that are known to be true (false) of this stereotype [Doyle, 1979]. These features are definitional of the stereotype. The label question marks features that are in but incomplete. In this case, (AVOIDANCE-GOAL (STATE)) refers to an unknown goal that needs to be filled in when the information comes in. This is represented as a goal with an unknown goal-object. Finally, the label hypothesized marks features that were true in this story but were not causally relevant to the explanation. These features are retained for the purposes of recognition and learning. Since AQUA does not assume that its explanations are complete, there is the possibility of learning more about this explanation in the future that would help to determine whether these features have explanatory significance. This has not yet been implemented in AQUA.

The stereotype is used to index the new explanation in memory (figure 2). After reading this story, AQUA uses the new stereotype to retrieve the blackmail explanation when it reads other stories about Lebanese teenagers going on suicide bombing missions.

This stereotype is built through generalization under causal constraints from the hypotheses that were considered, including the ones that were ultimately refuted. The causal constraints are derived both from the successful explanation (blackmail) as well as from unsuccessful hypotheses, if any (here, religious fanaticism).

5.2.1 Learning from successful explanations

Clearly, much of stereotype.79 comes from the motivational aspects of the blackmail explanation. AQUA retains those goals, goal orderings and beliefs of the character in the story that are causally implicated in the blackmail explanation. Since blackmail relies on a goal ordering between two goals, one of which is sacrificed for the other, the stereotype must specify that the character has a goal that he or she values above p-life. The stereotype also specifies that the character would normally not have the goal of performing terrorist missions, since this is part of the blackmail explanation. In the above story, AQUA infers the following goals and goal-orderings for the actor (corresponding to (1), (2) and (3) of xp-blackmail, page 2):

```
Building new stereotype (STEREOTYPE.79):

    Inferring GOALS
    from XP-BLACKMAIL (successful):
        THE ACTOR WANTED TO PRESERVE HIS OWN LIFE.
        THE ACTOR DID NOT WANT TO PERFORM THE
            TERRORIST MISSION.
        THE ACTOR WANTED TO AVOID SOMETHING.

    Inferring GOAL-ORDERINGS
    from XP-BLACKMAIL (successful):
        THE GOAL OF THE ACTOR TO AVOID SOMETHING
            WAS MORE IMPORTANT THAN THE GOAL OF
            THE ACTOR TO PRESERVE HIS OWN LIFE.
```

These goals and goal-orderings are added to the stereotype being built. At this point, stereotype.79 has the following features:

```
Typical goals:
    P-LIFE (in)
    A-DESTROY (OBJECT) (out)
    AVOIDANCE-GOAL (STATE) (question)
Typical goal-orderings:
    AVOIDANCE-GOAL (STATE) over P-LIFE (question)
```

5.2.2 Learning from failed explanations

Many explanation-based learning programs learn only from positive examples (e.g., [Mooney and DeJong, 1985; Segre, 1987]). However, it is also possible to apply this technique to learn from negative examples (e.g., [Mostow and Bhatnagar, 1987; Gupta, 1987]). AQUA uses refuted hypotheses to infer features that should *not* be present in the newly built stereotype. These are features which, if present, would have led to the hypothesis being confirmed.

For example, in the blackmail story, AQUA knows that the person being blackmailed is not a religious fanatic, since the religious fanatic explanation, which depended on this fact, has been refuted. The kind of person likely to be blackmailed into suicide bombing is, therefore, not a religious fanatic.[5] This feature is recorded in the newly built stereotype.

```
Building new stereotype (STEREOTYPE.79):

    XP-RELIGIOUS-FANATIC failed because:
        THE BOY DID NOT BELIEVE FANATICALLY IN THE
        SHIITE MOSLEM RELIGION.

    Inferring BELIEFS
    from XP-RELIGIOUS-FANATIC (failed):
        THE ACTOR DID NOT BELIEVE FANATICALLY IN
        A RELIGION.
```

This results in a new belief being added to stereotype.79:

[5] As before, this is a stereotypical inference and not a logically correct one. A religious fanatic could indeed be blackmailed into suicide bombing; however, on reading a story about a religious fanatic going on a suicide bombing mission, blackmail would not normally come to mind. This means that xp-blackmail-suicide-bombing should not be indexed under religious-fanatic, at least on the basis of this example.

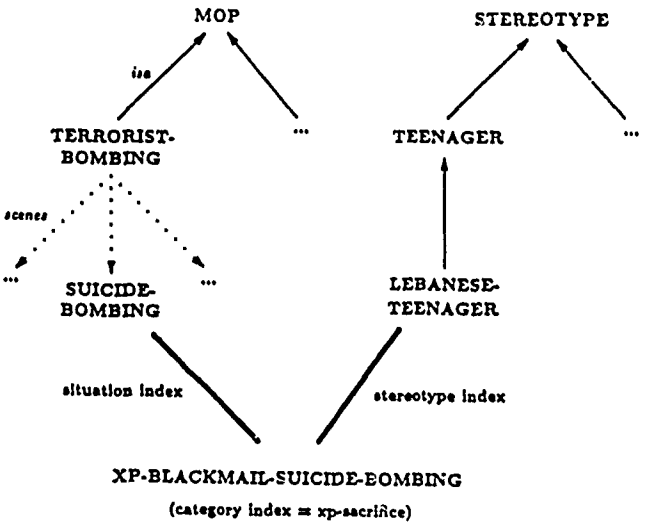


Figure 2: Learning stereotype indices for XPs. Upward lines represent `isa` links, and downward dotted lines represent scenes of MOPs. Heavy lines represent indices to XPs. Here, AQUA has just built a stereotype index from `stereotype.79`, representing a `lebanese-teenager`, to `xp-blackmail-suicide-bombing`.

```
Typical beliefs:
    RELIGIOUS-ZEAL = NOT A FANATIC (in)
```

The reason that learning from the failed explanation works in this example is that the blackmail explanation specifies that the person being blackmailed would normally not have the goal to perform that action. This rules out other explanations which would result in this goal. Our theory does not deal with the issue of multiple successful explanations; more research needs to be done in this area.

6 Modifying existing explanation patterns

6.1 Associating new questions with XPs

Suppose AQUA reads the blackmail story with only the religious fanatic XP for suicide bombing in memory. When reading this story, AQUA is handed an explanation for the suicide bombing: the story explicitly mentions that the bomber was blackmailed. In a sense, then, the story has been understood since an explanation for the bombing has been found. However, one could not really say that AQUA had understood the story if it didn't ask the question, *What could the boy want more than his own life?* Unless this question is raised while reading the story, one would have to say that AQUA had missed the point of the story.

Such questions correspond to gaps in the explanation structures that are built during the understanding process (figure 3). These questions are associated with the XP, and may be answered later in the story or when the XP is applied to a future story. When they are answered, the understander can elaborate and modify the XP, thus achieving a better understanding of the causality represented by the XP.

6.2 Incremental refinement of XPs by answering questions

In addition to raising new questions, of course, an understander must answer the questions that is already has in order to improve its knowledge of the domain. AQUA uses its questions to focus the understanding process, and learns when these questions are answered.

For example, consider the following story:

> JERUSALEM — A young girl drove an explosive-laden car into a group of Israeli guards in Lebanon. The suicide attack killed three guards and wounded two others. ...
>
> The driver was identified as a 16-year-old Lebanese girl. ... Before the attack, she said that a terrorist organization had threatened to harm her family unless she carried out the bombing mission for them. She said that she was prepared to die in order to protect her family.

When this story is read, AQUA retrieves the new `xp-blackmail-suicide-bombing` and applies it to the story. The question that is pending along with this explanation is also instantiated. When the question is answered, it is replaced by a new node representing the `protect-family` goal, and becomes part of `xp-blackmail-suicide-bombing`. Since no explanations are known for the newly added node, this in turn becomes a new question about the elaborated XP (not shown in the figure). The question is seeking a reason for the unusual `goal-ordering` of the `actor`, in which `protect-family` is given a higher priority than `p-life`.

When the elaborated XP is applied to a new suicide bombing story, the new node will now be one of the premises of the hypothesis, causing AQUA to ask

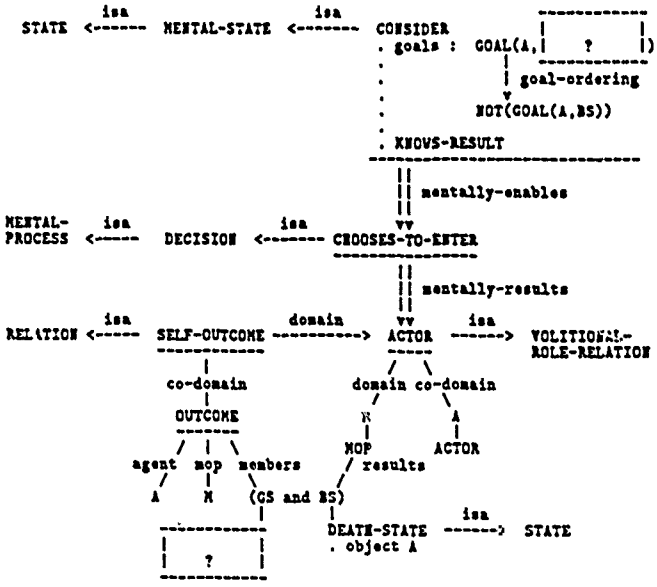


Figure 3: Associating new questions with XPs. The XP represents a situation in which an agent A volitionally performs (chooses-to-enter) an action whose outcome is known (knows-result) to be the death-state of A, as well as an unknown state that A wants more than he wants to avoid his death-state (the goal-ordering). The unknown goal represents the new question, *What could the actor want more than his own life?* This is depicted as an empty box, representing a gap in the program's knowledge. The XP is elaborated by filling in this gap when this question is answered.

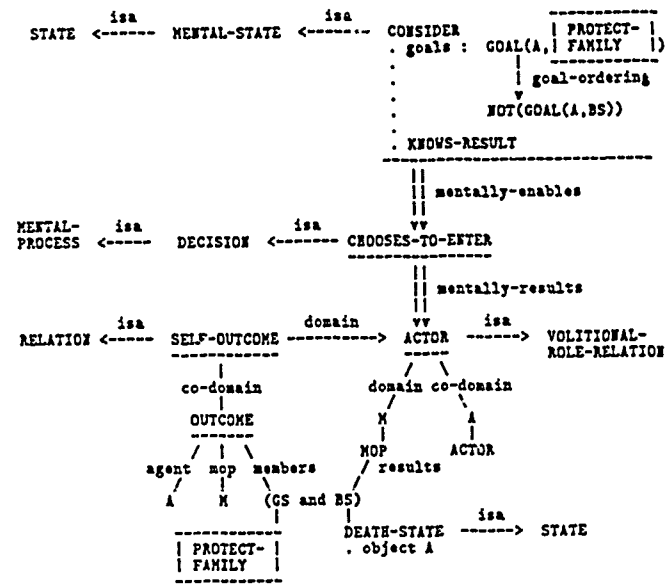


Figure 4: Elaborating an XP through incremental learning. The changed portion is depicted as a newly filled-in box, representing the answering of the question that was indexed at that point (compare with figure 3).

whether the actor was trying to protect his family. This reflects a deeper understanding of this particular scenario and is shown in figure 4. The new question will also be instantiated, causing AQUA to look for an explanation for the unusual **goal-ordering**. Should new questions be raised and then answered during future stories, AQUA will again be able to elaborate this XP in a similar manner. Thus AQUA evolves a better understanding of the "blackmailed into suicide bombing" scenario through a process of question asking and answering.

7 Conclusions

Explanation patterns are used for constructing explanations for anomalous situations by applying stereotypical packages of causality from similar situations encountered earlier. Thus XPs are abstract cases that are used as paradigmatic examples of stereotypical situations.

This paper presents a theory of XP learning through the incremental modification of existing XPs, using explanation-based learning techniques to constrain the modification process. The modifications involve the adaptation and elaboration of XPs, as well as the learning of indices for XPs. Both types of knowledge are essential in any case-based reasoning system. The theory is implemented in the AQUA program, which learns about terrorism by reading newspaper stories about unusual terrorist incidents in the Middle East.

References

[DeJong and Mooney, 1986] G. F. DeJong and R. J. Mooney. Explanation-Based Learning: An Alternative View. *Machine Learning*, 1(2):145–176, 1986.

[Diettrich and Michalski, 1981] T. G. Diettrich and R. S. Michalski. Inductive Learning of Structural Descriptions: Evaluation Criteria and Comparative Review of Selected Methodologies. *Artificial Intelligence*, 16:257–294, 1981.

[Doyle, 1979] J. Doyle. A Truth Maintenance System. *Artificial Intelligence*, 12:231–272, 1979.

[Flann and Dietterich, 1989] N. S. Flann and T. G. Dietterich. A Study of Explanation-Based Methods for Inductive Learning. *Machine Learning*, 4:187–226, November 1989.

[Gupta, 1987] A. Gupta. Explanation-Based Failure Recovery. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 606–610, Seattle, WA, July 1987. AAAI.

[Hammond, 1989] K. J. Hammond, editor. *Proceedings of a Workshop on Case-Based Reasoning.* Morgan Kaufmann, Inc., Pensacola Beach, FL, May 1989.

[Kolodner, 1988] J. L. Kolodner, editor. *Proceedings of a Workshop on Case-Based Reasoning.* Morgan Kaufmann, Inc., Clearwater Beach, FL, May 1988.

[Mitchell *et al.*, 1986] T. M. Mitchell, R. Keller, and S. Kedar-Cabelli. Explanation-Based Generalization: A Unifying View. *Machine Learning*, 1(1):47–80, 1986.

[Mooney and DeJong, 1985] R. J. Mooney and G. F. DeJong. Learning Schemata for Natural Language Processing. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 681–687, Los Angeles, CA, August 1985. IJCAI.

[Mostow and Bhatnagar, 1987] J. Mostow and N. Bhatnagar. FAILSAFE – A Floor Planner that uses EBG to Learn from its Failures. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 249–255, Milan, Italy, August 1987. IJCAI.

[Ram, 1987] A. Ram. AQUA: Asking Questions and Understanding Answers. In *Proceedings of the Sixth Annual National Conference on Artificial Intelligence*, pages 312–316, Seattle, WA, July 1987. American Association for Artificial Intelligence, Morgan Kaufman Publishers, Inc.

[Ram, 1989] A. Ram. *Question-driven understanding: An integrated theory of story understanding, memory and learning.* Ph.D. thesis, Yale University, New Haven, CT, May 1989. Research Report #710.

[Schank and Ram, 1988] R. C. Schank and A. Ram. Question-driven Parsing: A New Approach to Natural Language Understanding. *Journal of Japanese Society for Artificial Intelligence*, 3(3):260–270, May 1988.

[Schank, 1986] R. C. Schank. *Explanation Patterns: Understanding Mechanically and Creatively.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[Segre, 1987] A. M. Segre. *Explanation-Based Learning of Generalized Robot Assembly Tasks.* Ph.D. thesis, University of Illinois at Urbana-Champaign, Urbana, IL, January 1987. Technical Report UILU-ENG-87-2208.

[Wilensky, 1978] R. Wilensky. *Understanding Goal-Based Stories.* Ph.D. thesis, Yale University, Department of Computer Science, New Haven, CT, 1978.

EXPLANATION-BASED AND EMPIRICAL LEARNING

Integrated Learning in a Real Domain

F. Bergadano, A. Giordana, L. Saitta

Universita` di Torino, Corso Svizzera 185,
Torino, Italy Tel: +39 11 771.2002

D. De Marchi, F. Brancadori

SOGESTA s.p.a., Urbino, Italy

Abstract

This paper describes the results obtained in applying the learning system ENIGMA to a fault diagnosis problem of electromechanical devices at ENICHEM (Ravenna, Italy). The system ENIGMA is capable of learning structured knowledge from examples and a domain theory, using an integrated inductive/deductive paradigm.

The results are compared with the ones obtained by an expert system, designed for the same task, in which the knowledge base was acquired using the traditional method of expert interview. The comparison indicates that performances obtained by the learning system are systematically better than the ones obtained by the manually developed expert system. The conclusion is that, even if still leaving room for improvements, automated learning is a viable approach to the construction of expert systems, from the point of view of both obtainable performance and of limiting the development time and cost.

1. Introduction

It is widely recognized that the feasibility of expert systems exhibiting human like performances strongly depends upon the possibility of developing mechanisms for automating the processes of knowledge acquisition and maintenance. In the last decade, a number of research projects have been devoted to machine learning and knowledge acquisition. Although many steps ahead have been made, especially for the problem of learning concept descriptions from examples, we have still to recognize that the problem is, in general, extremely hard. This is confirmed by the fact that a number of learning systems have been described in the literature, but very few real applications were addressed, in which machine learning proved to be capable of generating a knowledge base with the same (or better) performance as the one constructed by a human expert. We mention, in this sense, the results obtained by Michalski in developing automatic classification systems for agricultural [7] and medical [8] applications. In domains where the learning events can be represented as vectors of <attribute, value> pairs, interesting results have been obtained using decision trees [11,12].

Many of the above mentioned results in classification systems have been obtained using mainly inductive methods; however, an important requirement, which characterizes many applications, is that the learned rules be understandable in the light of a pre-existing knowledge of the domain; this is particularly true for diagnostic systems.

This paper describes the work done, and the results obtained, in a pilot project aimed at checking the real possibilities offered by the state of the art in Machine Learning in order to automate the process of knowledge base construction for an expert system oriented to electromechanical troubleshooting. In order to achieve these goals, the prototype system ENIGMA, based on an integrated inductive/deductive paradigm [4], has been developed and an extensive experimentation has been performed, the most important aspects of which will be described in the following. The performances of the knowledge base acquired by ENIGMA have been compared with those of MEPS, a rule based Expert System, whose knowledge has been manually acquired by interviewing the domain expert [5]. However, performances are not the only useful parameter for the comparison: also knowledge understandability and meaningfulness and development time have been taken into consideration. The results have been considered encouraging enough to justify a large funding, by ENI, for a new project aimed at developing an industrial version of this learning system.

2. The Learning Problem

The case study has been supplied by the Enichem-Anic chemical plant at Ravenna, Italy. In this plant a technique of predictive maintenance is applied to a large set of apparata including motor-pumps, turbo-alternators and ventilators.

All these apparata share the common feature of possessing a rotating shaft to which various rotors are connected. When a machine possesses rotating

elements, several unavoidable vibratory motions are induced in its parts; these vibrations occur also during the correct machine operation and are not dangerous as long as their amplitude remains limited. When some fault occurs in the machine, new, anomalous vibrations appear, beside other manifestations. The aim of the predictive maintenance is to locate failures (still in the initial stage) and to diagnose their severity, through an analysis of these vibrations which is called *mechanalysis*. Mechanalysis basically performs a Fourier analysis of the vibratory motions taken in prespecified and labelled points, precisely on the supports of the machine components. By means of a special *analyzer*, the technician obtains, for each support, the amplitude and velocity of the global vibration along the vertical, horizontal and axial direction; furthermore, the same data can be taken for each of the harmonic components of the vibrations. Also qualitative evaluation of the vibration phase can be done.

Mechanalysis has strong mathematical foundations in vibration theory, and, hence, the relationships between anomalous frequencies and faults could be, in principle, predicted. In practice, things are not so simple, as, usually, many more vibrations than those predicted occur; this is due to several reasons, such as mechanical imperfections in the parts, mutual influence among motions, resonance phenomena and fault co-occurrence. Moreover, a vibration does not begin abruptly, but its intensity increases over time, until a level, deemed to be dangerous, may be reached.

The proposed learning task was that of learning automatically from examples of mechanalysis, and with the help of background knowledge, a knowledge base suitable to derive diagnoses of the type produced by human experts. This task can be seen as a classical case of learning concept descriptions [6], but shows many difficulties which are not present in other learning tasks described in the literature, residing both in the kind of available data and in the conceptualization of the problem.

First of all, the examples are complex, each one consisting, for the motor-pumps, of about 20 to 60 measures taken in differents points and conditions of the machine. Each measure has 2 or 3 attributes (value, direction and frequency, if appropriate). But, more than that, the examples are very noisy; in fact all the measures are affected by large uncertainty margins, depending both on the intrinsic limits of the measurement apparata and on the human subjectivity in recording the observed values.

From the conceptual point of view, the principal difficulty resides in the fact that the expert's conclusion mainly arises from a global evaluation of the mechanalysis measures. a particular frequency pattern (or value) may acquire great relevance in a given context and may not be significant in another. Moreover, only few of the many measures are important for that particular diagnosis and the human expertise consists exactly in knowing how to identify them. This "globality" characteristic makes it difficult to define an adequate description language. It turns out, in fact, that the single mechanalysis measures are too low level as features and cannot be used directly to build up a description space. On the contrary, features of a higher level, defined in terms of groups of items, are to be introduced in order to describe hypotheses. This form of "constructive" learning [10] has been strongly guided, in the current implementation, by the domain theory.

3. The Learning System

The system ENIGMA is basically an evolution of an earlier version, the system ML-SMART [2,3], enhanced in order to include deductive capabilities as described in [4]. In fact, several attempts to apply to the described case study the former version of ML-SMART, which was a purely inductive system, generated knowledge that was very difficult to understand in the light of the existing domain theory.

We will not describe here the system ENIGMA, being a detailed description already available in [1,3,5], but we will mention some points necessary to understand how the system has been applied.

ENIGMA receives in input a set of learning events and a body of background knowledge described as a Horn theory and produces in output a structured knowledge base of classification rules. The peculiarity that makes this system suitable to deal with structured domains is that the learning events are described as vectors of items. Each item is in turn a vector of <attribute,value> pairs and corresponds to a part (subpattern) of a concept instance.

In the present case the learning events correspond to the mechanalysis data, obtained through Fourier analysis, which are collected in a table of the type described in Fig. 1.

Inside the table the data are arranged into groups of three rows; each group corresponds to a given support and each row to one spatial direction (Horizontal, Vertical and Axial). A first group of two columns (denoted by "Total Vibration") contains the measures of amplitude and velocity of the total vibration, whereas a second group (denoted by "Fourier Analysis") contains the measures of frequency, velocity and (possibly) phase of the harmonic components of the vibration. Notice that, for the harmonics, the measure of the velocity v (for which more reliable analyzers exist) allows the amplitude to be known as well, being this last proportional to v through the (known) w. The measure consists, in some cases, of a single value (the index of the analyzer is stable), whereas, in others, of a range (the analog index of the analyzer osciliates between two extremes). This distinction is an important factor for the differential diagnosis. The qualitative behavior (stable, unstable, oscillating, rotating, fixed) of the vibration phase, when observed, is denoted by a letter attached to the corresponding frequency value. For instance, the "i" occurring in the figure denotes an instable phase.

Support	Direction	Total Vibration		Fourier Analysis				
		Amplitude [μm]	Speed [mm/s]	ω [CPM]	v [mm/s]	.....	ω [CPM]	v [mm/s]
A	Hor	7/11	2.4/2.6	3000 i	0.7/0.9	.....	18K	0.7
	Vert	4/8	1.2/1.4	3000	0.2/0.7	.....	18K	0.4
	Ax	20	12	3000	3/3.2	.....	18K	0.8/1

Fig. 1: Organization of the data collected during a mechanalysis.

A mechanalysis table is described to ENIGMA by supplying an item for each non empty entry. Each item is described by a vector of attributes characterizing the support, the direction, the amplitude, the type (total vibration or Fourier Analysis), the value of the measure, the "normal" value, and the rotation speed of the shaft. These attributes correspond to what in Explanation Based Learning are called "operational" predicates (elementary features). Other predicates (higher level features) can be defined using a Horn theory. In particular, it is possible to let ENIGMA work by applying pure EBG [9], if a complete theory, defining a non operational description of the concept, is given. However, this is practically difficult to achieve and ENIGMA was provided with a theory that only defines high level features capturing contextual information.

The rules learned by ENIGMA take the general form:

$$r: \quad H_j;\ \varphi \xrightarrow{w} H_k \vee H_l \qquad (1)$$

Rule r can be interpreted as follows. Suppose that H_0 is the set of all classes and $h \in H_0$ is the concept to be identified in a given event f. Suppose moreover, that, owing to some reasoning made by using other rules (of this type), we arrived at supposing that h can only belong to the subset $H_j \subset H_0$; then, if the assertion φ is verified on f, the rule (1) concludes that h belongs to H_k with probability w or to H_l with probability 1-w. The relations $H_k \cap H_l = \emptyset$ and $H_j \supset H_k, H_l$ always hold. H_j is called the *context* of the rule, H_k the *primary implication* and H_l the *secondary implication*. If the probability is w=1, the secondary implication is not present. As a special case, the set H_k may consist of a single concept h.

The assertion φ is a first order logic formula, expressed with operational predicates only. Numerical quantifiers such as Atleast n, Atmost n, Exactly n and negation are also allowed. Usually the primary implication of a rule coincides with the context of another formula. Formulas having the same primary implication are implicitely considered as OR-ed. As a consequence, the knowledge base learned by ENIGMA can be described as a graph of rules. ENIGMA produces such a graph by searching in the rule space using a general-to-specific strategy, starting from the most general formula true$\rightarrow H_0$ and generating more and more specific formulas until classification rules are discovered. This process is guided by statistical heuristics which trades off consistency and completeness criteria and by the background knowledge supplied at the beginning. The strategies and the heuristics are described in [2,3,4,5].

4. The Learning Set

All the experiments have been performed using a set F_0 of N=209 mechanalysis tables (examples), filled by an experienced domain expert and referring to diagnoses of motor-pumps.

The considered faults can be grouped into six classes:

C_1 = Problems in the joint
C_2 = Faulty bearings
C_3 = Mechanical loosening
C_4 = Basement distortion
C_5 = Unbalance
C_6 = Normal operation conditions

However, as mentioned in Section 2, these faults rarely occur in isolation and, even then, it is not always possible to individuate them precisely. Thus, not all the N examples have been univoquely classified by the human expert, but, on the contrary, the diagnoses generated by him followed the taxonomy reported in Fig. 2. According to this diagnostic taxonomy, the following intermediate classes have been used by the expert:

C_7 = Shaft misalignment ($C_7 = C_1 \cup C_4$)
C_8 = Problems in the pump ($C_8 = C_2 \cup C_3 \cup C_5$)
C_9 = Problems in the motor ($C_9 = C_2 \cup C_3 \cup C_5$)
C_{10} = Problems in the machine ($C_{10} = C_8 \cup C_9$)

The *ambiguity* $\alpha(f)$ of a classified example f is the minimum number of classes f is hypothesized to belong to; for instance, an example f classified in class C_8 has an ambiguity $\alpha(f) = 3$. Moreover, a diagnosis H_i will be said *more specific* that a diagnosis H_j iff H_j is an ancestor of H_i in the diagnostic taxonomy.

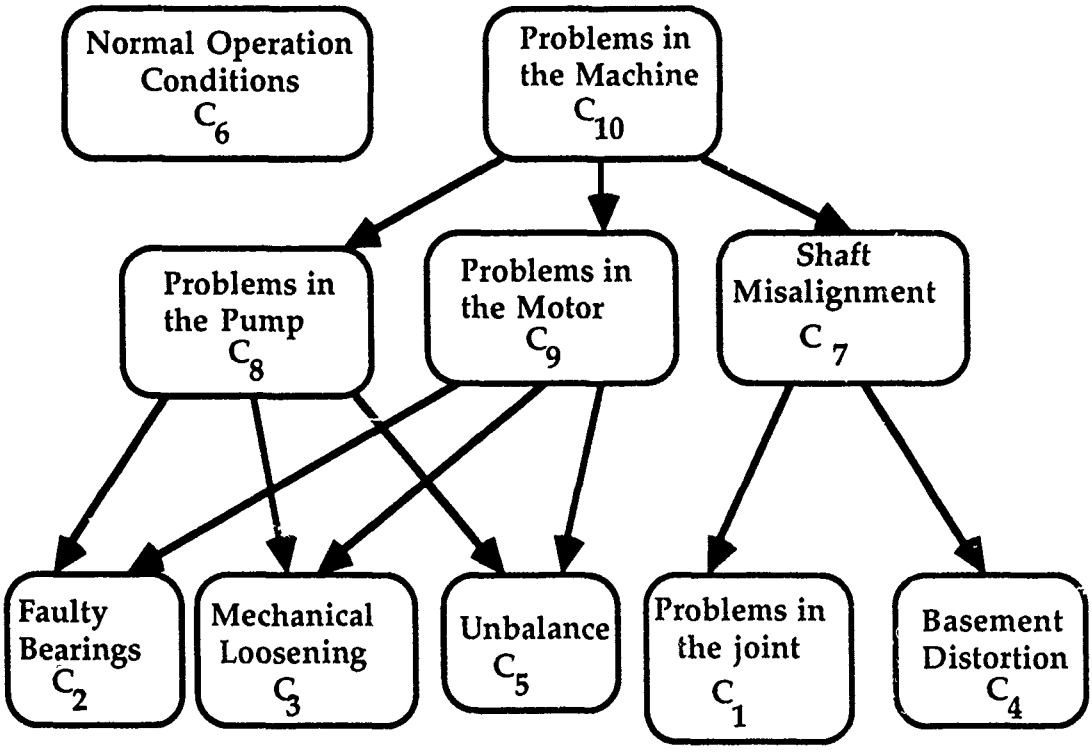


Fig. 2 - Diagnostic taxonomy of the motor-pump faults

Table I
Diagnoses generated by the expert. The classes correspond to the taxonomy in Fig. 2

Class	No. of Examples	Ambiguity	Class	No. of Examples	Ambiguity
C_{10}	29	5	$C_7 \ll C_5$	2	1
C_7	42	2	$C_7 \ll C_2$	2	1
C_8	26	3	$C_1 \ll C_3$	2	1
C_9	8	3	$C_1 \ll C_8$	1	1
C_1	13	1	$C_7 \ll C_3$	1	1
C_2	23	1	$C_1 \ll C_2$	4	1
C_3	6	1	$C_1 \ll C_9$	1	1
C_4	5	1	$C_4 \ll C_9$	1	1
C_5	13	1	$C_4 \ll C_8$	1	1
C_6	27	1	$C_3 \ll C_4$	1	1
			$C_1 \ll C_5$	1	1

Obviously, we desire that the automated system be at least as specific as the expert was.The ambiguity parameter α roughly corresponds to the amount of efforts required to exactly individuate the cause of a single fault. In fact, the higher α, the greater the number of components that need to be examined. For example, if *faulty bearings* in the pump is assessed, then only the bearings have to be disassembled; if, instead, only *problems in the pump* can be hypothesized, then the whole pump has to be dismounted.

In Table I the expert's classification of the N examples is reported; the average ambiguity of the examples is $\alpha = 2.08$. Notice that, whereas an internal node of the diagnostic taxonomy denotes uncertainty about the right choice among the descendant nodes, the intersection $C_i \cap C_j$ denotes co-occurrence of both C_i and C_j; for this reason, the ambiguity of the diagnosis $C_i \cap C_j$ is evaluated as the minimum between those assigned to C_i and C_j. Regarding the expert's classification, it has to be pointed out that, in many cases, the expert was actually able to generate a less ambiguous hypothesis than the one reported in Table I. However, he judged this more specific diagnosis as out of reach for a system not acquainted with a deeper understanding of the domain; then, he indicated what, in his opinion, was an acceptable answer for a prototype automated system. Moreover, the generation of a more precise diagnosis, by the part of the expert, entrained an error rate of about 5% (according to the expert's subjective estimate), whereas the diagnoses reported in Table I are the most specific ones, which are still error free. In this context, we say that an *error* occurred when the true class is not included in the set of the proposed ones.

5. Results with the Expert System MEPS

MEPS is a prototype expert system [6] developed manually by means of interviews with the same domain expert who supplied the classified examples. MEPS' knowledge is represented by means of rules and frames and contains both diagnostic and structural information. The chosen implementation environment is the GOLD-WORKS shell on an IBM Personal Computer. The system contains about 290 diagnostic rules and 70 structural frames. Its representation language is a first order logic based language with an associated continuous-valued semantics. The process of designing and implementing the MEPS prototype took about 18 months, 12 of which devoted to the acquisition, encoding and maintainance of the knowledge base.

In Table II the results obtained from the expert system MEPS are reported. MEPS performs, on a given case, an evidential reasoning and generates, as a result, a list of possible faults, ordered according to decreasing value of evidence. The recognition rate has been evaluated in two ways, a pessimistic and an optimistic one.

Table II
Results of MEPS on the examples of the set F_0

Ambiguity	Number of Cases	The best hypothesis was the correct one	The correct hypothesis was included in the proposed set
1	131	122	122
2	60	49	59
3	18	15	18

In the pessimistic case, only the best scored hypothesis h_{best} has been considered; if h_{best} is a descendant, in the diagnostic taxonomy, of the node corresponding to the diagnosis given by the expert and is an ancestor of the correct diagnosis, then h_{best} is considered correct (the system gives an answer of equal or higher specificity then that of the expert and conatins the correct class). In the optimistic case, MEPS' diagnosis is considered correct if at least one hypothesis with the preceding characteristics is included in the set of generated hypotheses. The obtained average ambiguity is α=1.46, the pessimistic error rate is η=0.86 and the optimistic error rate is η^*=0.95.

6. Results Obtained Using ENIGMA

The learning experiments with ENIGMA were performed exploiting the incremental abilities of the system and consisted of a sequence of six runs (phases).

In the first phase, 60 cases, randomly chosen, were used as learning set (LS) and the remaining 149 examples as test set (TS). Afterwards, 20 examples,

randomly selected among the 149, were used to update the current knowledge base and the 129 left over ones acted as test set. Also the error rate on the (60+20) training examples has been computed Notice that the previously used 60 examples are *test* examples for the knowledge acquired with the 20 following ones.The whole process has been repeated 5 times, by randomly choosing the examples to be added in each phase, but keeping their number fixed, and the average results are reported in Table III. As an example, we report here one of the rule learned by ENIGMA:

"If the shaft rotating frequency is w_0 and the harmonic at w_0 is reported to have high intensity and the harmonic at $2w_0$ is reported to have high intensity in at least two measurements, then the example is an instance of one of the classes C_1, C_4 or C_5".

Table III
Results of the automatic system ENIGMA with incremental learning

	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	Phase 5
Number of added training examples	60	20	24	21	20	22
Cardinality of the Test Set (TS)	149	129	105	84	64	42
Recognition rate Learning Set (LS)	0.97	0.99	0.96	0.98	0.97	0.96
Recognition rate Test Set	0.93	0.94	0.93	0.9	0.9	0.86
Recognition rate complete set	0.94	0.95	0.93	0.95	0.94	0.94
Ambiguity	1.15	1.24	1.18	1.18	1.23	1.19
Number of rules in the Knowledge Base	39	70	128	131	142	147

7. Discussion

An interesting comment about the results of the automatic learning is that the performance is quite stable by varying the number of seen examples and eventually slightly degrades. This counter-intuitive phenomenon is due to the fact that, because of the ambiguity inherent in the examples, adding new ones to the training set does not contribute any new information, but, on the contrary, mixes up infor[illegible] from different faults. In fact, by denoting by α_j the average ambiguity of the learning examples at j-th phase, we notice from Table IV an increase confusion of this set, which is responsible for the above mentioned phenomenon.

Table IV
Ambiguity of the learning set in each learning phase. The last column refers to examples of classes C_1-C_5.

	Ambiguity in the added Learning Set	Ambiguity in the global Learning Set	Ambiguity of the examples of classes C1-C5
Phase 1	1.98	1.98	2.18
Phase 2	2.23	2.04	2.24
Phase 3	2.36	2.11	2.31
Phase 4	2.23	2.13	2.33
Phase 5	2.85	2.23	2.43
Phase 6	2.36	2.25	2.44

Table V
Comparison between ENIGMA and MEPS

	Ambiguity	Recognition rate on complete set	Recognition rate on test set	Development time
ENIGMA	1.21	0.95	0.94	18 months
MEPS	1.46	0.95	-----	4 months

More precisely, the main confusion arises among the five fault classes, whereas the examples in class C_6 (non faulty machine) are always good examples; in fact, both the expert system MEPS and the knowledge generated by ENIGMA never confuse among faulty and non faulty machines. As it is not possible to select more "clean" examples than those used in these experimentations, the only way to further increase the KB performances seems to be that of supplying both MEPS and ENIGMA with a deep model of the domain, allowing a more complex, but more subtle, reasoning to be performed, as the human expert does. The evolution of the approach toward this direction is under development and preliminary encouraging results have already been obtained [5,13,14].

A second point to be investigated is the comparison between the knowledge base acquired by ENIGMA and that of MEPS. Some parameters useful to this aim are reported in Table V.

For ENIGMA, the knowledge base acquired during the second phase has been actually retained and used. As one can see from Table V, the performance of the two systems are comparable, but the knowledge automatically acquired needed much less time and efforts to be built up. Notice that, even if we must consider, in general, the recognition rate on the test set as the performance measure, it is more fair, in this case, to compare the two systems on the basis of the recognition rate on the complete set of 209 examples. Even though this discussion could seem pointless, given the substantial identity of the two values (0.94 versus 0.95) and the fact that in the second phase the learning events are only 80 over 209, it is interesting from a methodological point of view. In fact, the considered 209 cases (spanning a time period of about 25 years) are the very source of the knowledge of the human expert who supplied the MEPS rules. He did not have a teacher, nor there was expertise available in advance, and he formed his expertise by handling this set of cases (according to his own statement). Then, MEPS knowledge base is in fact tested on its own learning set. It is also interesting to notice that about 1/3 of the rules acquired by ENIGMA was coincident with corresponding rules in MEPS.

For what concerns the development time, an initial phase of problem mastering was common to both projects (about 2 months) and a further month was spent in preparing and memorising the data. Afterward, the manual knowledge acquisition and updating lasted 12 months, whereas ENIGMA acquired the knowledge base in a few hours.

To make the acquired knowledge more understandable to the expert, a domain theory, defining higher level features, has been given to the system. An example of rules in this domain theory is the following:

"If the shaft rotating frequency is ω_0 and a vibration has frequency ω and ω is a multiple of ω_0, then ω is a HARMONIC of ω_0"

The process of defining and implementing this theory tooks about one month more.

8.Conclusions

In this paper we described an application of the learning system ENIGMA to a real problem of mechanical troubleshooting. The problem was a difficult one, due to the complexity and high degree of noise of the data and to the effort required for choosing a suitable description language and a problem conceptualization.

The results obtained indicate that it starts to be realistic to apply machine learning techniques to significant problems, largely reducing the development time of expert systems, at the same time keeping a performance level comparable with that of expert systems developed with classical methodologies.

References

[1] F. Bergadano, F. Brancadori, A. Giordana, and L. Saitta. A system that learns diagnostic knowledge in a database framework. In *Proc. of the IJCAI89 workshop on knowledge discovery in databases*, Detroit, Michigan, 89.

[2] F. Bergadano, R. Gemello, A. Giordana, and L. Saitta. Smart: a problem solver for learning from examples. *Fundamenta Informaticae*, 12:29–50, 1989. Elsevier, Amsterdam.

[3] F. Bergadano, A. Giordana, and L. Saitta. Automated concept acquisition in noisy environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(4):555–577, 1988. New York.

[4] F. Bergadano and A. Giordana. A knowledge intensive approach to concept induction. In *Proc. of the Fifth Int. Conf. on Machine Learning*, pages 305–317, Morgan Kaufmann, Ann Arbor, MI, 1988.

[5] F. Brancadori and S. Radicchi. Acquisizione automatica della conoscenza. *Informatica Oggi*, 9(53):81–93, 1989.

[6] R. S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20:111–161, 1983.

[7] R. S. Michalski and R. L. Chilauski. Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition. *Int. Journal of Policy Analysis and Information Systems*, 4(2), 1980.

[8] R. S. Michalski, J. Hong, N. Lavrac, and I. Mozetic. The aq15 inductive learning system: an overview and experiments. In *Proc. of the First Int. Meeting on Advances in Learning*, Les Arcs, France, 1986.

[9] T. Mitchell, R. M. Keller, and S. Kedar-Cabelli. Explanation based generalization: a unifying view. *Machine Learning*, 1:47–80, 1986.

[10] S. Muggleton. Machine invention of first order predicates by inverting resolution. In *Proc. of the Fifth Int. Conf. on Machine Learning*, pages 339–352, Ann Arbor, MI, 1988.

[11] R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[12] B. Cestnik, I. Kononenko, and I. Bratko. Assistant 86: a knowledge elicitation tool for sophisticated users. In *Proc. of the 2nd european working sessions on learning*, Bled, Yugoslavia, 1987.

Incremental Version-Space Merging

Haym Hirsh
Computer Science Department
Rutgers University
New Brunswick, NJ 08903

Abstract

This paper describes a generalization of Mitchell's version-space approach to concept learning that significantly extends its range of applicability. The key idea is to remove the central idea of a version space being the set of concepts *strictly consistent* with training data, and allow arbitrary sets of concepts, however generated, as long as they can be represented by boundary sets. Learning is accomplished with version space intersection, rather than the traditional candidate-elimination algorithm. Applications of the learning method, *incremental version-space merging*, include learning from forms of inconsistent data and combining empirical and analytical learning.

1 Introduction

Concept learning can be viewed as a problem of search [Simon and Lea, 1974; Mitchell, 1978; 1982]—to identify some concept definition out of a space of possible definitions. Mitchell [1978] formalizes this view of *generalization as search* in his development of *version spaces*. He defines a version space to be the set of all concept definitions in a prespecified language that correctly classify training data—the positive and negative examples of the unknown concept. Although a landmark work, it was limited in its underlying assumption that the desired concept definition will be strictly consistent with all the given data. This work generalizes the version-space approach to concept learning to partially overcome this assumption.

The paper begins with a brief review of the version-space approach to concept learning, followed by an overview of the generalized approach. Summaries of two different applications of the resulting learning method are then presented: emulating and extending the candidate-elimination algorithm, and combining empirical and analytical learning. A third application, learning from data with bounded inconsistency, is presented elsewhere in these proceedings [Hirsh, 1990b]. The paper concludes with a discussion of computational issues for the approach.

2 Version Spaces and the Candidate-Elimination Algorithm

Given a set of training data and a language in which the desired concept must be expressed (which defines the space of possible generalizations concept learning will search), Mitchell [1978] defines a version space to be "the set of all concept descriptions within the given language which are consistent with those training instances". Mitchell noted that the generality of concepts imposes a partial order that allows efficient representation of the version space by the boundary sets S and G representing the most specific and most general concept definitions in the space. The S- and G-sets delimit the set of all concept definitions consistent with the given data—the version space contains all concepts as or more general than some element in S and as or more specific than some element in G.

Given a new instance, some of the concept definitions in the version space for past data may no longer be consistent with the new instance. The *candidate-elimination algorithm* manipulates the boundary-set representation of a version space to create boundary sets that represent a new version space consistent with all the previous instances plus the new one. For a positive example the algorithm generalizes the elements of the S-set as little as possible so that they cover the new instance yet remain consistent with past data, and removes those elements of the G-set that do not cover the new instance. For a negative instance the algorithm specializes elements of the G-set so that they no longer cover the new instance yet remain consistent with past data, and removes from the S-set those elements that mistakenly cover the new, negative instance. The unknown concept is determined when the version space has only one element, which in the boundary set representation is when the S- and G-sets have the same single element.

To demonstrate the candidate-elimination algorithm, consider a robot manipulating objects on an assembly line. Occasionally it is unable to grasp an object. The learning task is to form rules that will allow the robot to predict when an object is graspable. To make the example simple, the only features of objects that the robot can identify, and hence the only features that may appear in the learned rules, are shape and size. An object may be shaped like a cube, pyramid, octahedron, or sphere, and may have large or small size. Further structure to the shape attribute may be imposed by including in the robot's vocabulary the term "polyhedron"

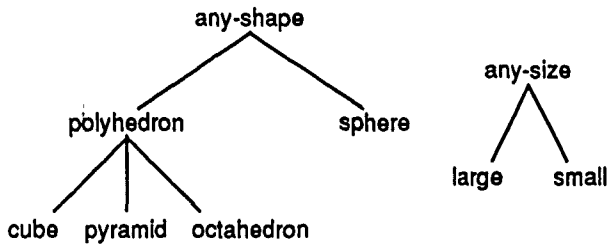


Figure 1: Generalization Hierarchies.

for cubes, pyramids, and octahedra. The generalization hierarchies that result are shown in Figure 1. Concept definitions take the form

```
Size(X,small) ∧ Shape(X,polyhedron)
    → Graspable(X),
```

which will be abbreviated to "[small, polyhedron]." The language is assumed to be sufficient for expressing the desired concept, and the data are assumed to be consistent.

The first object the robot tests is graspable, and is thus a positive example of the target concept. It is a small cube, and hence the initial version space has boundary sets S={[small, cube]} and G={[any-size, any-shape]}. The second object on the assembly line cannot be grasped, so is a negative instance. It is a small sphere, yielding new boundary sets S={[small, cube]} and G={[any-size, polyhedron]}—the only way to specialize the G-set to exclude the new instance but still cover the S-set element is to move down the generalization hierarchy for shape from any-shape to polyhedron. A further negative instance, a large octahedron, prunes the version space yet more, to S={[small, cube]} and G={[any-size, cube]; [small, polyhedron]}. The new G-set now has two elements since there are two ways to specialize the old G-set to exclude the new instance but still cover the S-set element. After a final, positive instance that is a small pyramid, the boundary sets become S=G={[small, polyhedron]}, yielding the final generalization that all small polyhedral objects are graspable.

3 Incremental Version-Space Merging

There were two principal insights in Mitchell's work. The first was to consider and keep track of a set of candidate concept definitions, rather than keeping a single definition deemed best thus far. The second insight was that the set of all concept definitions need not be explicitly enumerated and maintained, but rather the partial ordering on concepts could be exploited to provide an efficient means of representation for the space of concept definitions. The key idea in this work is to maintain Mitchell's two insights, but remove its assumption of strict consistency with training data—a version space is generalized to be any set of concept definitions in a concept description language representable by boundary sets.

Given the generalized definition of version spaces, the candidate-elimination algorithm can no longer be used—it, too, assumes strict consistency with data. Thus an alternative incremental learning method was developed. Rather than basing the learning algorithm on shrinking the version space, the new algorithm is instead based on version-space intersection. Given a version space based on one set of information, and another based on a second set of information, the intersection of the two version spaces reflects the union of the sets of information. Such version-space intersection forms the basis for the incremental learning method, called *incremental version-space merging*, developed as part of this work.

The algorithm for computing the intersection of two version spaces is called the *version-space merging algorithm*. It computes the intersection using only boundary-set representations. This is pictured in Figure 2. Given version space VS_1 with boundary sets S_1 and G_1, and VS_2 with boundary sets S_2 and G_2, the version-space merging algorithm finds the boundary sets $S_{1\cap2}$ and $G_{1\cap2}$ for their intersection, $VS_1 \cap VS_2$ (labeled $VS_{1\cap2}$). It does so in a two-step process. The first step assigns the set of minimal generalizations of pairs from S_1 and S_2 to $S_{1\cap2}$, and assigns the set of maximal specializations of pairs from G_1 and G_2 to $G_{1\cap2}$. The second step removes overly general elements from $S_{1\cap2}$ and overly specific elements from $G_{1\cap2}$. In more detail:[1]

1. For each pair of elements s_1 in S_1 and s_2 in S_2 generate their most specific common generalizations. Assign to $S_{1\cap2}$ the union of all such most specific common generalizations of pairs of elements from the two original S-sets. Similarly, generate the set of all most general common specializations of elements of the two G-sets G_1 and G_2 for the new G-set $G_{1\cap2}$.
2. Remove from $S_{1\cap2}$ those elements that are not more specific than some element from G_1 and some element from G_2. Also remove those elements more general than some other element of $S_{1\cap2}$ (generated from a different pair from S_1 and S_2). Similarly remove from $G_{1\cap2}$ those elements that are not more general than some element from each of S_1 and S_2, as well as those more specific than any other element of $G_{1\cap2}$.

The only information a user must give for this version-space merging algorithm to work is information about the concept description language and the partial order imposed by generality. The user must specify a method for determining the most general common specializations and most specific common generalizations of any two concept definitions. The user must also define the test of whether one concept definition is more general than another. Given this information about the concept description language, the two-step process above will intersect two version spaces, yielding the boundary-set representation of the intersection.

The result of this change of perspective, from version-space shrinking to version-space intersection, is that each new constraint that should reduce the version space of viable concept definitions must be represented as a version space itself, to be intersected with the current version space of candidates. When new information is obtained, incremental version-space merging forms the version space of

[1] Mitchell [1978] provides an equivalent version-space intersection algorithm.

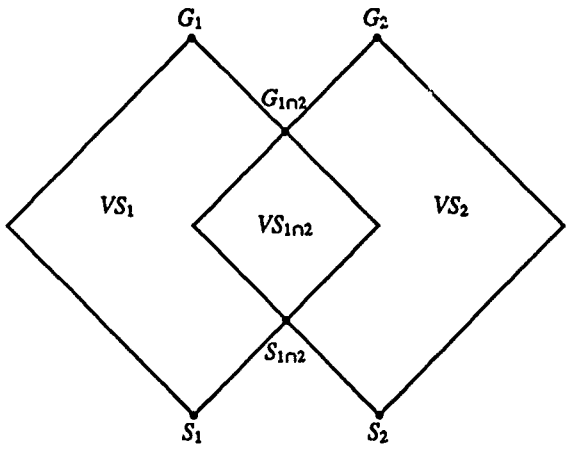


Figure 2: Version-Space Merging.

concept definitions that are relevant given the new information and intersects it with the version space for past information. This is pictured in Figure 3. As each new piece of information is obtained, its version space is formed (VS_I) and intersected with the version space for past data (VS_n) to yield a new version space (VS_{n+1}), which will itself be intersected with the version space for the next piece of information.

The general algorithm proceeds as follows:

1. Form the version space for the new piece of information.
2. Intersect this version space with the version space generated from past information.
3. Return to the first step for the next piece of information.

The initial version space contains all concept descriptions in the language, and is bounded by the S-set that contains the empty concept that says nothing is an example, and the G-set that contains the universal concept that says everything is an example.

The insight from which the generality of incremental version-space merging arises is that the specific learning task should define how each piece of information is to be interpreted—what the version space of relevant concept definitions should be. Use of incremental version-space merging requires a specification of how the individual version spaces should be formed in the first step for each iteration. Forming the version space of concept definitions consistent with each instance (i.e., those concept definitions that correctly classify the given instance) results in an emulation of the candidate-elimination algorithm, and permits an extension that handles cases of ambiguous data (such as when the color attribute of some instance is known to be either brown or black without knowing which). Forming the version space containing concept definitions consistent with the explanation-based generalization of data provides a way to integrate empirical and analytical learning. These are the applications discussed in this paper. Further details are presented elsewhere [Hirsh, 1989b; 1989a; 1990a].

A further application of incremental version-space merging, to learn from inconsistent data, is presented elsewhere in these proceedings, and will be briefly summarized here. The approach taken is to forego a solution to the full problem of learning from inconsistent data, and instead solve a subcase, called bounded inconsistency. Data are said to have bounded inconsistency when some small perturbation to the description of any bad instance will result in a good instance (such as when misclassifications are due to small measurement errors). When this is true, a learning system can search through the space of concept definitions that correctly classify either the original data, or small perturbations of the data. Instance version spaces will contain all concept definitions consistent with either the instance or some neighboring instance description. The resulting version space after each incremental intersection not only contains concept definitions that correctly classify all the training data (if any such definitions exist), but also those that miss some (or even all) of the data by only a small amount. Further details are presented elsewhere [Hirsh, 1989b; 1990b].

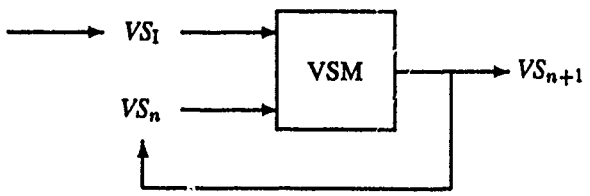


Figure 3: Incremental Version-Space Merging.

4 The Candidate-Elimination Algorithm: Emulation and Extensions

Incremental version-space merging should maintain the functionality of the original version-space approach. This section demonstrates how the candidate-elimination algorithm can be emulated using incremental version-space merging, and furthermore describes an extension that enables learning from ambiguous data (such as when features are not uniquely identified).

4.1 Emulating the Candidate-Elimination Algorithm

The key idea for emulating the candidate-elimination algorithm with incremental version-space merging is to form the version space of concept definitions strictly consistent with each individual instance and incrementally intersect these version spaces with incremental version-space merging. The results after each incremental intersection can be shown to be the same as those after each step of the candidate-elimination algorithm [Hirsh, 1989b].

Emulating the candidate-elimination algorithm with incremental version-space merging in this manner requires forming the version space of concept definitions consistent with a single instance in boundary-set representation. This is done as follows. If the training instance is a positive example, its S-set is assigned the most specific elements in the language that include the instance. When the single-representation trick holds (i.e., for each instance, there is a concept definition whose extension is only that instance [Dietterich *et al.*, 1982]), the S-set contains the instance as its sole element. When it does not hold, the learning-task-specific method that generates instance version spaces must determine the set of most specific concept definitions in the language that cover the instance. The new G-set contains the single, universal concept that says that everything is an example of the concept. If the training instance is a negative example, its S-set is taken to be the single, empty concept that says that nothing is an example of the concept, and its G-set is the set of minimal specializations of the universal concept that don't cover the instance. This forms the boundary-set representation of the version space of concept definitions consistent with a single training instance.

The emulation can be summarized as follows:

1. Form the version space of all concept definitions consistent with an individual instance.
2. Intersect this new version space with the version space for past data (which starts as the full version space containing all concepts in the concept description language) to generate a new version space.
3. Return to Step 1 for the next instance.

Note that this merely instantiates the general incremental version-space merging algorithm given earlier, specifying how individual version spaces are formed. Furthermore, in contrast to the candidate-elimination algorithm, this emulation allows the first instance to be negative and does not assume the single-representation trick.

To demonstrate this incremental version-space merging implementation of the candidate-elimination algorithm, the robot learning task presented in Section 2 will again be used. The initial version space has boundary sets S={$\emptyset$} and G={[any-size, any-shape]} (where $\emptyset$ represents the empty concept that says nothing is an example of the target concept). The version space for the first positive instance, a small cube, has the boundary sets S={[small, cube]} and G={[any-size, any-shape]} (Step 1 of the incremental version-space merging algorithm), and when merged with the initial version space simply returns the instance version space (Step 2). This is obtained using the version-space merging algorithm (Section 3): in its first step the most specific common generalizations of pairs from the two original S-sets are formed—here it is the most specific common generalizations of $\emptyset$ and [small, cube]: {[small, cube]}; the second step prunes away those that are not minimal and those not covered by elements of the two original G-sets, but here nothing need be pruned. Similarly, for the new G-set the most general common specialization of [any-size, any-shape] and [any-size, any-shape] is [any-size, any-shape], and nothing need be pruned.

The version space for the second, negative example, a small sphere, is defined by S={$\emptyset$} and G={[large, any-shape]; [any-size, polyhedron]}—nothing more general excludes the negative instance. When merged with the previous version space, the new boundary sets are S={[small, cube]} and G={[any-size, polyhedron]}. This is obtained by taking for the new S-set the most specific common generalizations of [small, cube] and $\emptyset$ that are more specific than [any-size, any-shape] and one of [large, any-shape] and [any-size, polyhedron]—i.e., covered by elements of the two original G-sets. This simply yields {[small, cube]}. For the new G-set the most general common specializations of [any-size, any-shape] and [large, any-shape]—{[large, any-shape]}—and the most general common specializations of [any-size, any-shape] and [any-size, polyhedron]—{[any-size, polyhedron]}—are taken for the new G-set, but [large, any-shape] must be pruned since it is not more general than an element of one of the original S-sets.

The third, negative example, a large octahedron, has boundary sets S={$\emptyset$} and G={[small, any-shape]; [any-size, sphere]; [any-size, cube]; [any-size, pyramid]}. Merging this with the preceding boundary sets yields S={[small, cube]} and G={[any-size, cube]; [small, polyhedron]}. Finally, the last, positive instance, a small pyramid, has boundary sets S={[small, pyramid]} and G={[any-size, any-shape]}, resulting in the final version space S=G={[small, polyhedron]}.

4.2 Ambiguous Data

When an instance is not uniquely identified, it is said to be *ambiguous*. For example, only knowing a range for a person's height or age is a form of ambiguous data. More extreme examples are when data are provided at too general a level (such as only knowing that someone is tall), or when attributes are totally missing. The incremental version-space merging emulation of the candidate-elimination algorithm provides a mechanism for doing concept learning even when given ambiguous data. The basic idea is to form the version space of concept definitions for ambiguous data by identifying the set of all concept definitions consistent with any potential identity for the ambiguous instance; its version space should include concept definitions consistent with any possible interpretation of the instance. For example, if a positive instance is known to be either black or brown, its version space should contain all concept definitions that include the black case plus all concept definitions that include the brown case. This version space can be viewed as the union of two version spaces, one for black and the other for brown.

Defining a version space requires defining the contents of its boundary sets. For ambiguous training data this is done by setting the boundary sets to the most specific and general concept definitions consistent with some possibility for the training instance. If the instance is a positive example, the S-set contains the most specific concept definitions that include at least one possible identity for the ambiguous instance. If the single-representation trick holds, the S-set contains the set of all instances that the training instance might truly be. The G-set contains the universal concept that matches everything. If the instance is negative the S-set contains the empty concept that matches nothing and the G-set contains the minimal specializations of the universal concept that do not include *one* of the possibilities for the

uncertain data.

The algorithm can thus be summarized as follows:

1. (a) Form the set of all instances the given instance might be.
   (b) Form the version space of all concept definitions consistent with any individual instance in this set.
2. Intersect the version space with the version space for past data to obtain a new version space.
3. Return to Step 1 for the next instance.

This is again just the algorithm of Section 3, with a new specification of how individual version spaces are formed. If there is no ambiguity, the algorithm behaves like the candidate-elimination algorithm emulation above.

Note that ambiguous data cannot be handled through the use of internal disjunction [Michalski, 1983]. An example of an internal disjunction would be the concept definition [small, octahedron∨cube]. This says that small objects that are either octahedra or cubes will be positive. Both small octahedra *and* small cubes are included as positive by it. An ambiguous instance, on the other hand, cannot guarantee that both will be positive; it may be that only small cubes are positive, whereas the internal disjunction would errantly include small octahedra. A correct solution to handling ambiguous data must not rule out concept definitions whose extension only includes one of the possible identities of an ambiguous instance.

To demonstrate how ambiguous data are handled, the learning task of Section 2 is again used. If the first, positive instance were known to be small and either cube or octahedron, the instance version space would have boundary sets S={[small, cube]; [small, octahedron]} and G={[any-size, any-shape]}. After the second instance (a small sphere, negative example) is processed, the resulting boundary sets are S={[small, cube]; [small, octahedron]} and G={[any-size, polyhedron]}. The third instance (a large octahedron, negative example) results in S={[small, cube]; [small, octahedron]} and G={[any-size, cube]; [small, polyhedron]}. It takes the final instance (a small pyramid, positive example) to finally make the version space converge to S=G={[small, polyhedron]}.

5 Combining Empirical and Analytical Learning

The previous section described how incremental version-space merging can be used to emulate and extend the candidate-elimination algorithm. This section describes the use of incremental version-space merging to implement and extend Mitchell's [1984] proposal for combining empirical and analytical learning. The key idea is to form version spaces consistent with the results of explanation-based generalization (EBG) [Mitchell *et al.*, 1986], rather than consistent with ground data. The problem addressed is:

Given:

- *Training Data*: Positive and negative examples of the concept to be identified. Training data are expressed within an instance description language, whose terms are assumed to be operational.
- *Concept Description Language*: A language in which the final concept must be expressed. It is a superset of the instance description language, and is where generalization hierarchies would appear.
- *Positive-Data Domain Theory* (optional): A set of rules and facts for proving that an instance is positive. Proofs terminate in elements of the instance description language.
- *Negative-Data Domain Theory* (optional): A set of rules and facts for proving that an instance is negative. Proofs terminate in elements of the instance description language.

Determine:

- A set of concept definitions in the concept description language consistent with the data.

The method processes a sequence of instances as follows, starting with the first instance:

1. (a) If possible, apply EBG to the current instance to generate a generalized instance. Do so for all possible explanations. If no explanation is found, pass along the ground data.
   (b) Form the version space of all concept definitions consistent with the (perhaps generalized) instance. If there are multiple explanations include those concept definitions consistent with any single explanation.
2. Intersect this version space with the version space generated from all past data.
3. Return to the first step for the next instance.

This is again an instantiation of the general incremental version-space merging algorithm given earlier.

The basic technique is to form the version space of concept definitions consistent with the explanation-based generalization of each instance (rather than the version space of concept definitions consistent with the ground instance). The version space for a single training instance reflects the explanation-based generalization of the instance, representing the set of concept definitions consistent with all instances with the same explanation as the given instance. The merging algorithm has the effect of updating the version space with the many examples sharing the same explanation, rather than with the single instance. In this manner irrelevant features of the instances are removed, and learning can converge to a final concept definition using fewer instances.

The technique also applies to cases of multiple, competing explanations, when only one explanation need be correct. In such cases the version space of concept definitions consistent with one or more of the potential results of EBG is formed. EBG is applied to every competing explanation of an instance, each yielding a competing generalization of the instance. The space of candidate generalizations for the single instance contains all concept definitions consistent with *at least one* of the competing generalizations. The final generalization after multiple instances must be consistent with one of them. The situation is similar to that of ambiguous data (Section 4.2), only here it is unknown which

explanation is correct. Like the earlier treatment of ambiguous data, the version space contains all concept definitions consistent with at least one of the possibilities.

The version space of all concept definitions consistent with at least one explanation-based generalization of the instance is the union of the version spaces of concept definitions consistent with each individual explanation-based generalization. For positive examples this union has as its S boundary set the set of competing explanation-based generalizations, and the G boundary set contains the universal concept that labels everything positive. If one result of EBG is more general than another (e.g., one mentions a superset of the training-instance facts mentioned by the other), only the more specific result is kept in the S-set. Over multiple instances these version spaces consistent with the explanation-based generalizations are incrementally intersected to find the space of concept definitions consistent with the analytically generalized data.

The approach is also useful given theories for explaining *negative* data, when the system is provided with a theory capable of explaining why an instance is negative. For example, in search control an example of a state in which an operator should *not* be used is a negative instance, and a theory for explaining why the instance is negative would analyze why the instance is negative—that the operator does not apply, or that it leads to a non-optimal solution. This theory is then used to generalize the negative instance to obtain a generalization covering all instances that are negative for the same reason. Incremental version-space merging then uses this generalized instance by setting the S-set equal to the empty concept that says nothing is an example of the concept, and setting the G-set equal to all minimal specializations of the universal concept that do not cover the generalized negative instance. If there are multiple, competing explanations, the G-set contains all minimal specializations that do not cover at least one of the potential generalizations obtainable by EBG using one of the explanations.

Note that it is not necessary to have a complete theory capable of explaining (and generalizing) all correct instances for this technique to work. The version space of all concept definitions consistent with a plain non-generalized instance —whether negative or positive—can always be formed. Instead of using EBG, the version space consists of all concept definitions consistent with the instance, rather than its explanation-based generalization. If a theory for only explaining positive instances exists, negative instances can be processed without using EBG. If an incomplete theory exists (i.e., it only explains a subset of potential instances), when an explanation exists the version space for the explanation-based generalization of the instance can be used, otherwise the pure instance version space should be used. When there is no domain theory the learner degenerates to behave like the candidate-elimination algorithm. The net result is a learning method capable of exhibiting behavior at various points along the spectrum from knowledge-free to knowledge-rich learning.

To illustrate how incremental version-space merging combines empirical and analytical learning, two examples are presented. The first demonstrates how empirical learning generalizes beyond the specific results obtainable with EBG alone. The second demonstrates how empirical learning deals with multiple explanations.

5.1 Cup Example

The first example of the combination of incremental version-space merging and EBG demonstrates how a definition of Cup can be learned given incomplete knowledge about cups plus examples of cups. It is based on the examples given by Mitchell *et al.* [1986] and Flann and Dietterich [1990].

The following is the domain theory used (written in Prolog notation):

```
cup(X):-holds_liquid(X),
        can_drink_from(X),
        stable(X).
holds_liquid(X):-pyrex(X).
holds_liquid(X):-china(X).
holds_liquid(X):-aluminum(X).
can_drink_from(X):-liftable(X),
                   open_top(X).
liftable(X):-small(X).
stable(X):-flat_bottom(X).
```

It can recognize and explain some, but not all, cups. The concept description language used for this problem by empirical learning utilizes generalization hierarchies, including the knowledge that pyrex, china, and aluminum are nonporous materials, and that black and brown are dark colors. Note that this information is not present in the domain theory, but is known to be true in general. Empirical learning has many such possible generalizations. The goal for learning is to determine which potential generalizations, such as those mentioning nonporous material, are relevant.

Learning begins with the first, positive example:

```
china(cup1).
small(cup1).
open_top(cup1).
flat_bottom(cup1).
black(cup1).
```

EBG results in the rule

```
cup(X):-china(X),
        small(X),
        open_top(X),
        flat_bottom(X).
```

written "[china, small, open, flat, anycolor]" for short. This forms the S-set for the version space of the first instance (and the first step of incremental version-space merging), and its G-set contains the universal concept [anymaterial, anysize, anytop, anybottom, anycolor]. The second step of incremental version-space merging intersects this with the initial full version space, which gives back this first-instance version space.

Incremental version-space merging then returns to its first step for the next, positive instance, which is:

```
pyrex(cup2).
small(cup2).
open_top(cup2).
flat_bottom(cup2).
brown(cup2).
```

EBG results in the rule

```
cup(X):-pyrex(X),
        small(X),
        open_top(X),
        flat_bottom(X).
```

The S-set for this instance's version space contains the result of EBG, namely [pyrex, small, open, flat, anycolor], and its G-set contains the universal concept. Merging this with the version space for the first iteration yields a version space whose S-set contains [nonporous, small, open, flat, anycolor] and whose G-set contains the universal concept.

The final instance is a negative example:

```
aluminum(can1).
small(can1).
closed_top(can1).
flat_bottom(can1).
white(can1).
```

For this example the theory of negative data is assumed to include the following rules (among others):

```
not_a_cup(X):-
    cannot_drink_from(X).
cannot_drink_from(X):-
    closed_top(X).
```

EBG yields the following rule:

```
not_a_cup(X):-closed_top(X).
```

This is then used to determine the most general concept definitions that exclude this generalized case of not_a_cup, namely {[anymaterial, anysize, open, anybottom, anycolor]}, which forms the G-set of the version space for this third instance. The S-set contains the empty concept.

When this third instance version space is merged with the result of the previous two iterations of incremental version-space merging, the resulting S-set contains [nonporous, small, open, flat, anycolor] and the resulting G-set contains [anymaterial, anysize, open, anybottom, anycolor]. Note that the domain theories have done part of the work, with the color attribute being ignored and only the third attribute being deemed relevant for the negative instance, but empirical learning determining nonporous. Further data would continue refining the version space. However, it is already known that whatever the final rule, it will include small, nonporous, open-topped objects, like Styrofoam cups, which the original theory did not recognize as cups.

This simple domain also demonstrates the point made earlier about the technique degenerating to look like pure empirical learning. Consider the same examples, only without the domain theory present. At each iteration the resulting version space would be exactly the same as would be created by the candidate-elimination algorithm. The final version space would have an S-set containing [nonporous, small, open, flat, darkcolor], and a G-set containing two elements: [anymaterial, anysize, open, anybottom, anycolor] and [anymaterial, anysize, anytop, anybottom, darkcolor]. This version space contains more elements than the corresponding version space using EBG.

5.2 Can_put_on_table Example

As an example of using domain theories with multiple competing explanations consider the following domain theory for can_put_on_table:

```
can_put_on_table(X):-stable(X),
                     small(X).
can_put_on_table(X):-stable(X),
                     light(X).
stable(X):-flat_bottom(X).
```

It provides two potential explanations for when an object can successfully be placed on a table: the object must be stable, and either small or light. Given an unclassified instance, the theory cannot be used to predict its classification—whether it is positive or negative—since the theory can explain too many things, and will classify some potentially negative examples as positive. However, it *is* possible to explain a positive instance once its classification is given. The goal for learning is to determine a definition consistent with the data plus the subset of the theory that actually models the observed data. Furthermore, when there are multiple competing explanations of a given positive instance, later instances should allow determining which of the competing explanations is consistent across all data.

For example, given a can as a positive example of an object that can be placed on a table:

```
flat_bottom(can1).
small(can1).
light(can1).
```

the first step of the incremental version-space merging process uses EBG to form two rules, each corresponding to a different explanation:

```
can_put_on_table(X):-
    flat_bottom(X),
    small(X).
can_put_on_table(X):-
    flat_bottom(X),
    light(X).
```

These will be abbreviated to "[flat, small, anyweight]" and "[flat, anysize, light]". The resulting instance version space is bounded by an S-set containing these two concept definitions and a G-set containing the universal concept that says everything is an example of the concept. Intersecting this version space with the initial version space that contains all concept definitions in the concept description language yields the instance version space in return.

Returning to the first step of the learning process for the following positive, second instance,

```
flat_bottom(cardboard_box1).
big(cardboard_box1).
light(cardboard_box1).
```

EBG can only generate one rule:

```
can_put_on_table(X):-
    flat_bottom(X),
    light(X).
```

This results in an instance version space containing [flat, anysize, light] in the S-set and the G-set containing the universal concept. Merging the two instance version spaces (step two of incremental version-space merging) results in an S-set with the single element [flat, anysize, light] and the G-set containing the universal concept.

As an example of dealing with negative data with no negative-instance domain theory, consider the following negative example of can_put_on_table:

```
round_bottom(bowling_ball1).
small(bowling_ball1).
heavy(bowling_ball1).
```

The version space of concept definitions that do not include it has an S-set that contains the empty concept and a G-set that contains three concept definitions: {[flat, anysize, anyweight], [anybottom, large, anyweight], [anybottom, anysize, light]}. When merged with the version space for past data, incremental version-space merging yields a version space whose S set contains [flat, anysize, light] and whose G-set contains the two concept definitions [flat, anysize, anyweight] and [anybottom, anysize, light]. Subsequent data would further refine this version space.

6 Computational Complexity

Previous sections have described incremental version-space merging and two of its applications. This section analyzes the computational complexity of incremental version-space merging. Incremental version-space merging has two major steps: version-space formation, and version-space intersection. The computational complexity of each of these is first discussed, followed by an analysis of the complexity of the overall incremental version-space merging method. Further details are provided elsewhere [Hirsh, 1989b].

6.1 Version-Space Formation

The first step of each iteration of incremental version-space merging is to form the version space of concept definitions to consider given the current piece of information. Since this occurs at each step, it is clearly necessary for it to be feasible computationally. A general analysis of the complexity of this step is impossible—it depends both on the specific method for generating version spaces and on the particular concept description language being used. However, it is possible to do this analysis for specific approaches, and that is what will be done here. The particular method considered here works for conjunctive languages over a fixed set of k features, and handles the case when each version space is effectively the union of n version spaces (as occurs with ambiguous data, data with bounded inconsistency, and EBG with multiple explanations). The analysis applies to the applications given here, as well as for learning from data with bounded inconsistency [Hirsh, 1990b].

For positive instances, unpruned S-sets have n elements. Pruning nonminimal elements therefore requires at most n^2 comparisons of relative generality. Each comparison of concept definitions requires k feature comparisons. For tree-structured features, each feature comparison takes time proportional to the height h of the tree-structured hierarchy for that feature. However, h is at most $\log v$, where v is the maximum number of values any feature can take on.[2] Therefore the time to compare two concept descriptions takes time proportional to at most $k \log v$. Thus computing an S-set for tree-structured features takes at most time proportional to $n^2 k \log v$. In contrast, when features are ranges of the form $a \leq x < b$, feature comparisons take constant time, and thus computing the S-set takes time proportional to at most $n^2 k$. Either way, as long as forming the set of possible identities is tractable, handling positive data is tractable.

[2]Of course, this is only a useful bound on h when v is finite.

For negative instances there are nkb elements in the unpruned G-set, where b is the number of ways on average to specialize a feature to exclude a value.[3] Pruning nonmaximal elements therefore requires $(nkb)^2$ concept comparisons. For tree-structured features b is at most $(w-1)\log v$, where w is the maximum branching factor for all feature hierarchies, and thus computing the G takes time proportional to $(nk(w-1)\log v)^2 k \log v = (n(w-1))^2 (k \log v)^3$. In the case where features are ranges of the form $a \leq x < b$, with the range of x discretized to a fixed set of values (such as measuring values to the nearest millimeter [Hirsh, 1990b]), $b \leq 2$, and thus the time to compute a G-set is proportional to at most $n^2 k^3$. In both cases, this is again feasible as long as determining the n possibilities is tractable.

6.2 Version-Space Merging

The second step of incremental version-space merging is to intersect two version spaces using the version-space merging algorithm. The complexity of version-space merging is again dependent on the particular concept description language, and again the analysis is done for conjunctive languages over k features. The algorithm computes for the new S-set the most specific common generalization of pairs from the two S-sets that are covered by some element of each of the G-sets but not by some other element of the new S-set, and does a symmetrical process for the G-sets.

For both tree-structured features and features of the form $a \leq x < b$ the minimal common generalization of two concept descriptions is unique. Therefore, if there are m_1 and m_2 elements in the two initial S-sets, there are at most $m_1 m_2$ in the resulting unpruned S-set. When features are tree-structured, the process that computes the minimal generalization of two concept definitions takes time proportional to $k \log^2 v$. Computing the unpruned S-set thus takes time proportional to $m_1 m_2 k \log^2 v$. Computing minimal elements takes an additional $(m_1 m_2)^2 k \log v$, and removing elements not covered by some G-set element takes $m_1 m_2 (n_1 + n_2) k \log v$, where n_1 and n_2 are the two G-set sizes. Thus the overall complexity is proportional to at most $m_1 m_2 k \log^2 v + (m_1 m_2)^2 k \log v + m_1 m_2 (n_1 + n_2) k \log v$. The G-set case is nearly symmetric, with the exception that computing the maximal specialization of two concept definitions takes time proportional to at most $k \log v$, so the overall complexity is proportional to at most $n_1 n_2 k \log v + (n_1 n_2)^2 k \log v + n_1 n_2 (m_1 + m_2) k \log v$. Whichever is greater will be the overriding term.

When features are of the form $a \leq x < b$, computing the minimal generalization of two concept definitions takes time proportional to k. Computing the new S-set therefore takes time proportional to at most $m_1 m_2 k + (m_1 m_2)^2 k + m_1 m_2 (n_1 + n_2) k$, and for the G-set it is $n_1 n_2 k + (n_1 n_2)^2 k + n_1 n_2 (m_1 + m_2) k$. Again, whichever is greater is the overriding term.

6.3 Incremental Version-Space Merging

The preceding two subsections discussed the complexity of the individual steps taken during each iteration of incremen-

[3]If b is infinite (e.g., when the number of values v a feature may take is infinite), the use of version spaces inappropriate, since boundary-set sizes must be finite.

tal version-space merging. The complexity of the overall algorithm given a set of instances is a more difficult issue. It depends on the nature of the concept being learned, the concept description language, and the particular instances provided as data. In the worst case the process will be exponential in the number of instances, as has been pointed out by Haussler [1988] for the candidate-elimination algorithm, which is subsumed by this work.

However, in some cases where exponential growth could occur, it is possible to order the data so that exponential growth is avoided. One example where this is true (for consistent data) is in cases where after processing all the given data the resulting boundary sets are singleton. In such cases the data can always be ordered (in polynomial time) to guarantee that the boundary sets will remain singleton throughout learning. (The ordering algorithm basically processes positive data first, then processes negative "near-misses" [Winston, 1975] that remove selected "don't cares.") This is even true if Haussler's data set is a subset of the full data set. This is an area of current work.

Even when boundary sets do not grow exponentially in size there are techniques for further improving the performance of incremental version-space merging. Two have been explored: skipping data that do not change the version space, and selecting data that decrease boundary set size. The key idea in the first case is to note that instances that are classified the same by all members of the version space will not change the version space (assuming that the version space will not collapse to the empty set). This allows two improvements to incremental version-space merging: whenever one instance version space is a superset of another the first instance can be removed; and whenever the version space for the current instance is a superset of the current version space it can be skipped.

The second technique is based on the observation that, since the complexity of version-space merging is a function of boundary-set size, obtaining small boundary sets is a good idea. Furthermore, practice shows that once the boundary sets reach small size they typically stay small. Two specific heuristics were used that were generally successful in this task: selecting instances with small boundary sets, and selecting instances that have boundary sets with some overlap with the current version-space boundary sets.

7 Summary

This paper has presented a general framework for concept learning based on a generalization of Mitchell's version-space approach that removes its assumption of strict consistency with data. The observation on which incremental version-space merging is based is that concept learning can be viewed as the two-step process of specifying sets of relevant concepts and intersecting these sets. This observation is ultimately the major contribution of this work.

Acknowledgments

This paper summarizes part of the author's dissertation [Hirsh, 1989b], where more lengthy acknowledgments can be found. Among the people who have had an impact on the portion presented here are Bruce Buchanan, William Cohen, Tom Dietterich, Oren Etzioni, Nick Flann, David Haussler, Rich Keller, Tom Mitchell, Paul Rosenbloom, Paul Utgoff, Monte Zweben, and members of the GRAIL and Logic groups at Stanford and the MLRG group at CMU.

References

[Dietterich *et al.*, 1982] T. G. Dietterich, B. London, K. Clarkson, and G. Dromey. Learning and inductive inference. In P. Cohen and E. A. Feigenbaum, editors, *The Handbook of Artificial Intelligence, Volume III*. William Kaufmann, Los Altos, CA, 1982.

[Flann and Dietterich, 1990] N. S. Flann and T. G. Dietterich. A study of explanation-based methods for inductive learning. *Machine Learning*, 4(1), 1990.

[Haussler, 1988] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 26(2):177–221, Sept. 1988.

[Hirsh, 1989a] H. Hirsh. Combining empirical and analytical learning with version spaces. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 29–33, Ithaca, New York, 1989.

[Hirsh, 1989b] H. Hirsh. *Incremental Version-Space Merging: A General Framework for Concept Learning*. PhD thesis, Stanford University, June 1989.

[Hirsh, 1990a] H. Hirsh. Knowledge as bias. In D. P. Benjamin, editor, *Change of Representation and Inductive Bias*, pages 209–221. Kluwer, Boston, MA, 1990.

[Hirsh, 1990b] H. Hirsh. Learning from data with bounded inconsistency. In *Proceedings of the Seventh International Conference on Machine Learning*, Austin, Texas, June 1990.

[Michalski, 1983] R. S. Michalski. A theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 83–134. Morgan Kaufmann, Los Altos, CA, 1983.

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

[Mitchell, 1978] T. M. Mitchell. *Version Spaces: An Approach to Concept Learning*. PhD thesis, Stanford University, December 1978.

[Mitchell, 1982] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, March 1982.

[Mitchell, 1984] T. M. Mitchell. Toward combining empirical and analytic methods for learning heuristics. In A. Elithorn and R. Banerji, editors, *Human and Artificial Intelligence*. Erlbaum, 1984.

[Simon and Lea, 1974] H. Simon and G. Lea. Problem solving and rule induction. In H. Simon, editor, *Models of Thought*. Yale University Press, 1974.

[Winston, 1975] P. H. Winston. Learning structural descriptions from examples. In P. H. Winston, editor, *The Psychology of Computer Vision*, page Chapter 5. McGraw Hill, New York, 1975.

Average Case Analysis of Conjunctive Learning Algorithms

Michael J. Pazzani
Wendy Sarrett
Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717 USA
(pazzani@ics.uci.edu)
(sarrett@ics.uci.edu)

Abstract

We present an approach to modeling the average case behavior of learning algorithms. Our motivation is to predict the expected accuracy of learning algorithms as a function of the number of training examples. We apply this framework to a purely empirical learning algorithm, (the one-sided algorithm for pure conjunctive concepts), and to an algorithm that combines empirical and explanation-based learning. We evaluate the average-case models by comparing the accuracy predicted by the models to the actual accuracy obtained by running the learning algorithms.

1 Introduction

Most research in machine learning adheres to either a theoretical or an experimental methodology (Langley, 1989). Some attempt to understand learning algorithms by testing the algorithms on a variety of problems (e.g., Fisher, 1987; Minton, 1987). Others perform formal mathematical analysis of algorithms to prove that a given class of concepts is learnable from a given number of training examples (e.g., Valiant, 1984; Haussler, 1987). The common goal of this research is to gain an understanding the capabilities of learning algorithms. However, in practice, the conclusions of these two approaches are quite different. Experiments lead to findings on the average case accuracy of an algorithm. Formal analyses are typically deal with distribution-free, worst-case analyses. The number of examples required to guarantee learning a concept in the worst-case do not accurately reflect the number of examples required to learn an accurate concept in practice.

We have begun construction of an average case learning model to unify the formal mathematical and the empirical approaches to understanding the behavior of machine learning algorithms. Current experience with machine learning algorithms has lead to a number of empirical observations about the behavior of various algorithms. An average case model can explain these observations, make predictions, and guide the development of new learning algorithms.

1.1 PAC learning

Valiant (1984) has proposed a model to probabilistically justify the inductive leaps made by an empirical learning program. The probably approximately correct (PAC) model indicates that a system has learned a concept if the system can guarantee with high probability that its hypothesis is approximately correct. Approximately correct means that the concept will have an error no greater than ε (i.e., the ratio of misclassified examples to total examples is less than ε). The learning system is required to produce an approximately correct concept with probability $1-\delta$. For a given class of concepts, the PAC model can be used to determine an upper bound on the number of training examples required to achieve an accuracy of $1-\varepsilon$ with probability $1-\delta$. The PAC model has led to many important insights about the capabilities of machine learning algorithms. However, there is currently a wide gap between the theoretical results and the practical results of running learning algorithms on test data. In particular, Buntine (1989) has argued that the Valiant model can produce overly-conservative estimates of error and does not take advantage of information available in actual training sets.

1.2 FAC learning

Recently, Dietterich (1989) has proposed the frequently approximately correct (FAC) learning model. This learning model addresses the question of how frequently a learning algorithm acquires a hypothesis that is approximately correct on a training set of a given size. The frequency of

correctness is with respect to all possible training sets of the given size.

Dietterich has run three common learning programs on all 256 possible concepts of three binary features and found that best algorithm (the one-sided conjunctive learning algorithm, (Haussler, 1987)) can frequently (i.e, for 90% of the sets of training examples consisting of four of the eight possible examples) approximately (with accuracy greater than or equal to 87.5%) learn ten of these concepts. Dietterich has also calculated an upper-bound on the number of concepts that are FAC-learnable. For the parameters used in the experiments, at most 88 concepts are FAC-learnable. This implies that either the upper bound is too high, or the current generation of learning programs can be improved considerably.

1.3 Mathematical models of human learning

Several psychologists have created mathematical models of strategies proposed as models of human or animal learning (e.g., Atkinson, Bower & Crothers, 1965; Restle, 1958). These models have focused on average case behavior of learning algorithms. There are several reasons that these models cannot be directly applied to machine learning algorithms. First, these models typically study learning algorithms restricted to less complex concepts (e.g., single attribute discriminations) than those typically used in machine learning. Second, these models also address complications not present in machine learning programs since they must account for individual differences in human learners caused by such factors as attention, motivation, and memory limitations. Finally, the performance metric predicted by these models is generally the expected number of trials required to learn a concept with perfect accuracy. Typically, experimental studies of machine algorithms report on the observed accuracy on learning accuracy after a given number of examples.

2 The Average Case Learning Model

We have been developing a framework for average case analysis of machine learning algorithms. The framework for analyzing the expected accuracy of the hypothesis produced by a learning algorithm consists of determining:

- The conditions under which the algorithm changes the hypothesis for a concept.
- How often these conditions occur.
- How changing a hypothesis affects the accuracy of a hypothesis.

Clearly, the second requirement presupposes information about the distribution of the training examples. Therefore, unlike the PAC model, the framework we have developed is not distribution-free. Furthermore, to simplify computations (or reduce the amount of information required by the model) we will make certain independence assumptions (e.g., the probabilities of all irrelevant features occurring in training example are independent). Similarly, the third requirement presupposes some information about the test examples. We will make the same simplifying assumptions about the test examples as the training examples.

Determining conditions under which a learning algorithm changes a hypothesis requires an analysis of the operators used for creating and changing hypotheses. In this respect, the framework is more similar to the mathematical models of human and animal learning strategies than the theoretical results on machine learning algorithms which typically are concerned with the relationship between the size of the hypothesis space and the number of training examples. We will restrict our attention to learning algorithms that maintain a single hypothesis and incrementally modify the hypothesis when the hypothesis incorrectly classifies examples.

2.1 An average case model of wholist

We will first show how the framework can be applied to the wholist algorithm (Bruner, Goodnow, & Austin, 1956) a predecessor of the one-sided algorithm for pure conjunctive concepts (Haussler, 1987). Although this is a relatively simple algorithm, to our knowledge this is the first time an average case analysis of a machine learning algorithm has been shown to predict the expected accuracy in sufficient detail that it can be compared to observed accuracy obtained by running the algorithm. The goal of this analysis is to predict the probability that a randomly drawn example will be classified correctly by an algorithm.[1] The wholist algorithm is shown in Table 1. This algorithm incrementally processes training examples. The hypothesis maintained is simply the conjunction of all features that have appeared in all positive training examples encountered. The analysis will assume that the concept can in fact be represented as a conjunction of features. We will use the following notation in describing the algorithm:

1. Flann & Dietterich (1989) present an analysis of a component of IOE that is similar to wholist. However, their analysis calculates the number of training examples required to learn a hypothesis to a given accuracy.

Table 1. The wholist algorithm

```
1. Initialize the hypothesis to the conjunction of all features that
   describe training examples.
2. If the new example is a positive example, and the hypothesis
   misclassifies the new example, then remove all features from the
   hypothesis that are not present in the example.
3. Otherwise, do nothing.
```

- f_j is the j-th irrelevant feature of a training example. A feature is irrelevant if the true conjunctive definition of the concept does not include the feature.
- N is the number of examples (both positive and negative) seen so far.

The following information is required to predict the expected accuracy of wholist. Note that while this is much more information than required by the PAC model, this is exactly the information required to generate training examples to test the algorithm:

- P is the probability of drawing a positive training example.
- I is the number of irrelevant features.
- $P(f_j)$ is the probability that irrelevant feature j is present in a positive training example.

Note that the accuracy of this algorithm does not depend upon the number of relevant features that are conjoined to form the true concept definition. Therefore, the analysis does not make use of the total number of features or the number of relevant features.

The wholist algorithm has only one operator to revise a hypothesis. An irrelevant feature is dropped from the hypothesis when a positive training example does not include the irrelevant feature. Therefore, if i positive examples have been seen out of N total training examples, the probability that irrelevant feature j remains in the hypothesis (i.e., j has appeared in all i positive training examples) is $P(f_j)^i$.

The hypothesis created by the wholist algorithm misclassifies a positive test example if the hypothesis contains any irrelevant features that are not included in the test example. The wholist algorithm does not misclassify negative test examples (provided that the true concept definition can be represented as a conjunction of the given features). Therefore, the probability that feature j does not cause a positive test example to be misclassified after i positive training examples is given by $nm_{wholist}(f_j, i)$ (where nm stands for not misclassified):

$$nm_{wholist}(f_j, i) = 1 - (P(f_j)^i * (1 - P(f_j)). \quad [1]$$

If all irrelevant features are independent[2], then after i positive training examples, the probability that no irrelevant feature will cause a positive test example to be misclassified is:

$$\prod_{j=1}^{I} nm_{wholist}(f_j, i) \quad [2]$$

Finally, in order to predict the accuracy of the hypothesis produced by the wholist algorithm after N training examples, it is necessary to take into consideration the probability that exactly i of the N training examples are positive examples for each value of i from 0 to N. Therefore, the accuracy of the wholist algorithm (where accuracy is defined as the probability that a randomly drawn positive example will be classified correctly by an algorithm) can be given by:

$$\sum_{i=0}^{N} \left[b(i,N,P) * \prod_{j=1}^{I} nm_{wholist}(f_j, i) \right] \quad [3]$$

where $b(i,N,P)$ is the binomial formula: $b(i,N,P) = \binom{N}{i} * P^i (1 - P)^{(N-i)}$.

In Figure 1, the predicted and the actual accuracies of the hypothesis produced by the wholist algorithm are plotted. The mean accuracy of one hundred runs of the wholist algorithm is plotted as a function of the number of training examples. After every two training examples, the accuracy of current hypothesis was measured by classifying one hundred positive test examples. The concept to be learned was constructed from a set of ten features. Five of these features are irrelevant. The probability that a given irrelevant feature was present in a positive training example ranged from 5% to 30% (i.e., 0.05

2. If irrelevant features are not independent, then the average case learning model would also require conditional probabilities for the irrelevant features.

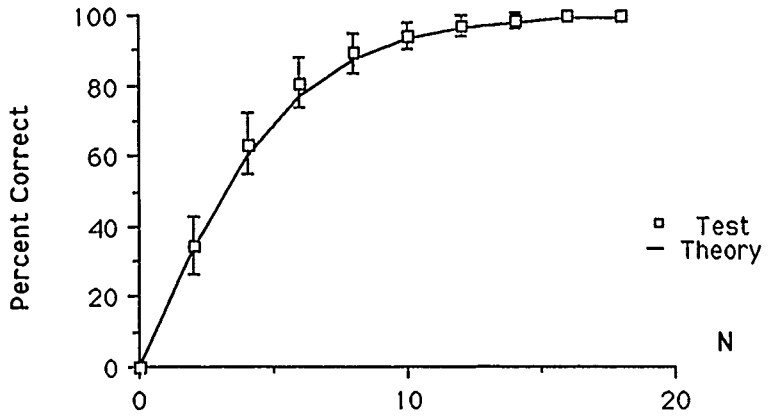


Figure 1: A comparison of the expected and actual accuracy of the wholist algorithm. The x-axis is the number of training instances and the y-axis is the percent of test instances correctly classified. The boxes represent the empirical means, the y-bars the 95% confidence interval around those means and the curve is the value predicted by Equation 3.

$\leq P(f_j) \leq 0.3$). In this simulation, the probability that a training example is a positive example is 40% (i.e., P = 0.4). The expected accuracy of wholist algorithm, as given by Equation 3 (under the conditions used to generate the training data) is plotted along with the results obtained by running the program.

In the following sections, we will illustrate how the framework we have developed for average case analysis of knowledge-free conjunctive learning algorithms can be used to gain insight on concept formation in the presence of background knowledge.

First, we describe a performance task and three learning algorithms that can be used to acquire the knowledge necessary to accomplish that task. Next, we present an average case analysis of each algorithm. Finally, in order to evaluate the average case analysis, we compare the predicted and observed accuracy of the algorithms on this task.

2.2 Learning a domain theory.

The learning and performance tasks to be analyzed differ from those commonly studied in machine learning. The difference is necessitated by the fact that we are interested in analyzing the use and learning of a domain theory for explanation-based learning (Dejong & Mooney, 1986; Mitchell, Keller, & Kedar-Cabelli, 1986). Learning a domain theory requires learning multiple concepts (one for each rule in the domain theory). The performance task is to infer if a predicate, p_2, is true in a world w given that a predicate p_1 is true. We will assume that the world can be represented as a set of binary features $x_1, x_2 \ldots x_n$. In this paper, variables starting with w will be used to refer to specific instances of worlds; variables starting with g will be used to refer to general descriptions of a class of worlds. We will assume that each g may be represented as a conjunction of a subset of the features used to describe specific instances of worlds.

Training examples are represented as specific instances of inference rules of the form[3] $p_1(w) \rightarrow p_2(w)$. The knowledge acquired by the learning system and used by the performance system is represented as inference rules of the form $p_1(g) \rightarrow p_2(g)$. These rules may be read as "if p_1 is true in a world g, then p_2 is true in g." g represents the set of conditions under which p_1 implies p_2. Inference rules of this form may be learned by generalizing all of the individual worlds in which p_1 implies p_2. A collection of such inference rules, learned from a variety of examples, may serve as the domain theory for explanation-based learning.

In this paper, we consider the case in which there are two possible means to determine if a

3. One possible situation in which training examples of this kind occur is in the induction of causal rules. In this case, a training of the form $p_1(w) \rightarrow p_2(w)$ may represent the situation in which an action p_1 was observed to occur in w and a teacher indicates that p_2 is an effect of p_1.

predicate c is true in world w when predicate a is true. The first is to acquire a rule $(a(G_{AC}) \rightarrow c(G_{AC}))$ that allows c to be inferred directly. The second is to acquire two rules $(a(G_{AB}) \rightarrow b(G_{AB})$ and $b(G_{BC}) \rightarrow c(G_{BC}))$ and allow the performance system to chaining rules to infer b from a, and then infer c from b. Note that G_{AC} can also be represented as $G_{AB} \wedge G_{BC}$. For example, a might represent striking an object, b might represent the object breaking and c might represent the owner of the object getting angry. The goal of the learning is to be able to predict when $a(G_{AC}) \rightarrow c(G_{AC})$: "If a person strikes an expensive fragile object, then the owner of the object will get angry." Two other rules may help in the prediction: "If a fragile object is struck, the object will break" and "If an expensive object breaks, the owner will be angry." Here, G_{AC} refers to the conditions "expensive and fragile", G_{AB} is the condition "fragile" and G_{BC} is the condition "expensive".

Three distinct groups of training examples are intermixed and presented incrementally to the learning system:

$a(W_{AC}) \rightarrow c(W_{AC})$

$a(W_{AB}) \rightarrow b(W_{AB})$

$b(W_{BC}) \rightarrow c(W_{BC})$

where W_{AC}, W_{AB} and W_{BC} are the sets of features of training examples. We will refer to the first type of training examples as *performance* examples since these examples will permit the learning system to learn a rule that directly enables the performance task. We will refer to the latter two types of examples as *foundational* examples because these examples do not allow the problem to be solved directly but provide a foundation for inferring when c is true.[4] We will assume that G_{AC}, G_{AB} and G_{BC} can be represented as pure conjunctive concepts. In this case, one means of learning G_{AC}, G_{AB}, and G_{BC} is to find the maximally specific conjunction of all examples of W_{AC}, W_{AB} and W_{BC}, respectively.

We will consider three related learning methods that can be used to acquire the knowledge for this performance task:

- wholist: The wholist method can be used to learn G_{AC}, from performance examples. Note that this method ignores the foundational examples.
- chaining: The wholist method can be used to learn G_{AB} and G_{BC} from foundational examples and c can be inferred from $a(G_{AB}) \rightarrow b(G_{AB})$ and $b(G_{BC}) \rightarrow c(G_{BC})$. Note that this method ignores the performance examples. This method can be viewed as learning the domain theory for explanation-based learning. However, it is irrelevant to the accuracy of the results of the learning whether the rule $(a(G_{AC}) \rightarrow c(G_{AC}))$ is cached by EBL (and updated whenever the domain theory is changed) or the performance task is solved by chaining.
- IOSC-TM (Sarrett & Pazzani, 1989a): The result of learning G_{AC} from performance examples, and the result obtained by learning G_{AB} and G_{BC} from foundational examples can be combined to form a composite hypothesis. This is the technique used by the integrated one-sided conjunctive learning algorithm with truth maintenance[5] (IOSC-TM). If a feature is n included in either the hypothesis found chaining or the hypothesis learned for G_{AC}, it is not included in the composite hypothesis. It is possible to combine the result of chaining and the result of learning G_{AC} empirically because each hypothesis only makes one-sided errors. The composite hypothesis formed by IOSC-TM is similar to the S set of the version space merging algorithm (Hirsh, 1989). However, unlike version-space merging, IOSC-TM learns the domain theory from foundational examples. Unlike the previous two algorithms, IOSC-TM can update its hypothesis when presented with performance or foundational examples.

Table 2 shows an example of the hypothesis produced by each of these three algorithms when run on the same training examples. Note that the hypothesis formed by IOSC-TM contains only those features that are in the hypothesis formed by wholist on performance examples, and in the hypothesis formed by chaining together foundational rules.

The analysis of the wholist algorithm presented in Section 2.1 will apply directly to this problem. Since wholist does not modify its hypothesis for G_{AC} when presented with foundational examples, P can

4. Note that the classification of a training example as a performance or foundational example is with respect to a specific performance task.

5. Truth maintenance implies that the composite hypothesis is updated immediately when a foundational example changes the hypothesis of a foundational rule. IOSC without truth maintenance waits until a performance example is misclassified before updating the composite hypothesis.

Table 2. Hypotheses produced by the three algorithms

Training Examples

$a(x_1=1, x_2=1, x_3=1, x_4=1, x_5=1) \rightarrow c(x_1=1, x_2=1, x_3=1, x_4=1, x_5=1)$
$a(x_1=1, x_2=0, x_3=1, x_4=1, x_5=1) \rightarrow b(x_1=1, x_2=0, x_3=1, x_4=1, x_5=1)$
$b(x_1=1, x_2=1, x_3=1, x_4=1, x_5=0) \rightarrow c(x_1=1, x_2=1, x_3=1, x_4=1, x_5=0)$
$a(x_1=1, x_2=1, x_3=0, x_4=0, x_5=1) \rightarrow b(x_1=1, x_2=1, x_3=0, x_4=0, x_5=1)$
$b(x_1=1, x_2=1, x_3=1, x_4=0, x_5=1) \rightarrow c(x_1=1, x_2=1, x_3=1, x_4=0, x_5=1)$
$a(x_1=1, x_2=1, x_3=1, x_4=0, x_5=0) \rightarrow c(x_1=1, x_2=1, x_3=1, x_4=0, x_5=0)$

Hypotheses

wholist: $a(x_1=1, x_2=1, x_3=1) \rightarrow c(x_1=1, x_2=1, x_3=1)$

chaining: $a(x_1=1, x_5=1) \rightarrow b(x_1=1, x_5=1)$
$b(x_1=1, x_2=1, x_3=1) \rightarrow c(x_1=1, x_2=1, x_3=1)$

(implicit) $a(x_1=1, x_2=1, x_3=1, x_5=1) \rightarrow c(x_1=1, x_2=1, x_3=1, x_5=1)$

IOSC-TM: $a(x_1=1, x_5=1) \rightarrow b(x_1=1, x_5=1)$
$b(x_1=1, x_2=1, x_3=1) \rightarrow c(x_1=1, x_2=1, x_3=1)$
$a(x_1=1, x_2=1) \rightarrow c(x_1=1, x_2=1)$

be viewed as the probability that a training example is a positive performance example and 1-P can be viewed as the probability that a training example is a foundational example. Note that we are interested in predicting the accuracy of the learning algorithms as a function of the total number of examples (both foundational and performance).

2.3 Average case analysis of using chaining

Chaining $a(G_{AB}) \rightarrow b(G_{AB})$ and $b(G_{BC}) \rightarrow c(G_{BC})$ requires using the wholist algorithm to learn two conditions (G_{AB} and G_{BC}). A positive test example will be correctly classified (i.e., for a given W_{AC} determining if $a(W_{AC}) \rightarrow c(W_{AC})$) if both G_{AB} and G_{BC} do not contain any irrelevant feature that is not present in the test example. Some new notation is necessary to express the probability that a positive test example is classified correctly by chaining:

- P_{AC}, P_{AB}, and P_{BC} are the probabilities of drawing a positive training example from $a(W_{AC}) \rightarrow c(W_{AC})$, $a(W_{AB}) \rightarrow b(W_{AB})$, and $b(W_{BC}) \rightarrow c(W_{BC})$, respectively. For brevity, here we only consider the case that $P_{AC} + P_{AB} + P_{BC} = 1$ (i.e., there are no negative training examples).

- $P_{AC}(f_j)$, $P_{AB}(f_j)$, and $P_{BC}(f_j)$ are the probabilities that irrelevant feature j is present in a positive training example from $a(W_{AC}) \rightarrow c(W_{AC})$, $a(W_{AB}) \rightarrow b(W_{AB})$, and $b(W_{BC}) \rightarrow c(W_{BC})$, respectively.

If i_1 positive training examples of $a(W_{AB}) \rightarrow b(W_{AB})$, and i_2 positive training examples of $b(W_{BC}) \rightarrow c(W_{BC})$ have been seen out of N total training examples, the probability that irrelevant feature j remains in the hypothesis is:

$$remain_{chaining}(f_j i_1, i_2) = 1 - ((1-P_{AB}(f_j)^{i_1}) * (1-P_{BC}(f_j)^{i_2})) \quad [4]$$

Therefore, the probability that feature j does not cause a positive test example to be misclassified is given by $nm_{chaining}(f_j, i_1, i_2)$:

$$nm_{chaining}(f_j, i_1, i_2) = 1 - (remain_{chaining}(f_j, i_1, i_2) * (1 - P_{AC}(f_j))) \quad [5]$$

If all irrelevant features are independent, then the probability that no irrelevant feature will cause a positive test example to be misclassified is:

$$\prod_{j=1}^{I} nm_{chaining}(f_j, i_1, i_2) \quad [6]$$

Finally, in order to predict the accuracy of the hypothesis produced by chaining after N training

$$\sum_{i_0}^{N}\sum_{i_1}^{N-i_0}\binom{N}{i_0,i_1,N-(i_0+i_1)}P_{AC}^{i_0}P_{AB}^{i_1}P_{BC}^{N-(i_0+i_1)} * \prod_{j=1}^{I} nm_{chaining}(f_j,i_1,N-(i_0+i_1)) \quad [7]$$

examples, it is necessary to take into consideration the probability that there are exactly i_0 positive training examples of $a(W_{AC}) \rightarrow c(W_{AC})$, i_1 positive training examples of $a(W_{AB}) \rightarrow b(W_{AB})$, and i_2 positive training examples of $b(W_{BC}) \rightarrow c(W_{BC})$ for each value of i_0, i_1 and i_2 from 0 to N. Note that because there are no negative training examples, i_2 is equal to $N-(i_0 + i_1)$. The multinomial formula is used to weight the value of $nm_{chaining}(f_j\, i_1,i_2)$ by the probability that various values of i_0, i_1 and i_2 occur. Therefore, the accuracy of the hypothesis produced by chaining can be given by Equation 7.

In Sarrett & Pazzani (1989b), we consider the general case in which the inference chain is of any given length and prove the general forms of the equations in this paper. In Section 2.5, we illustrate how well Equation 7 models the accuracy of the hypothesis produced by the chaining algorithm.

2.4 Average case of analysis of IOSC-TM

The IOSC-TM algorithm combines the hypothesis formed by chaining $a(G_{AB}) \rightarrow b(G_{AB})$ and $b(G_{BC}) \rightarrow c(G_{BC})$ and the hypothesis produced by wholist learning $a(G_{AC}) \rightarrow c(G_{AC})$. A positive test example will be incorrectly classified if it does not contain an irrelevant feature that meets both of the following conditions:

- Either G_{AB} or G_{BC} contains the irrelevant feature.
- G_{AC} contains the irrelevant feature.

If i_0 positive training examples of $a(W_{AC}) \rightarrow c(W_{AC})$, i_1 positive training examples of $a(W_{AB}) \rightarrow b(W_{AB})$, and i_2 positive training examples of $b(W_{BC}) \rightarrow c(W_{BC})$ have been seen out of N total training examples, the probability that irrelevant feature j remains in the hypothesis is:

$$remain_{IOSC}(f_j,i_0,i_1,i_2)=$$
$$P_{AC}(f_j)^{i_0} * (1-((1-P_{AB}(f_j)^{i_1}) * (1-P_{BC}(f_j)^{i_2}))) \quad [8]$$

Therefore, the probability that feature j does not cause a positive test example to be misclassified by IOSC-TM is given by $nm_{IOSC\text{-}TM}(f_j,i_0,i_1,i_2)$:

$$nm_{IOSC\text{-}TM}(f_j,i_0,i_1,i_2) =$$
$$1 - (remain_{IOSC}(f_j,i_0,i_1,i_2) * (1 - P_{AC}(f_j))) \quad [9]$$

In order to predict the accuracy of the hypothesis produced by chaining after N training examples, it is necessary to take into consideration the probability that there are exactly i_0 positive training examples of $a(W_{AC}) \rightarrow c(W_{AC})$, i_1 positive training examples of $a(W_{AB}) \rightarrow b(W_{AB})$, and i_2 positive training examples of $b(W_{BC}) \rightarrow c(W_{BC})$ for each value of i_0, i_1 and i_2 from 0 to N. As with chaining, i_2 is equal to $N-(i_0 + i_1)$. Therefore, the accuracy of the hypothesis produced by IOSC-TM can be given by Equation 10.

2.5 Evaluation of the average case model

In order to compare the accuracy of the hypotheses produced by the three algorithms under a variety of conditions, we substituted various values for $P_{AC}(f_j)$, $P_{AB}(f_j)$, $P_{BC}(f_j)$, P_{AC}, P_{AB}, and P_{BC} into Equations 3, 7 and 10. In addition, we ran each of the three algorithms on data generated according to the values of the parameters. Figure 2 shows the three algorithms when P_{AC} is 0.4, 0.2 and 0.1. In each case, P_{AB}, and P_{BC} are $(1 - P_{AC})/2$. The values of $P_{AC}(f_j)$, $P_{AB}(f_j)$, $P_{BC}(f_j)$ were randomly assigned for each feature from the range (0.01 to 0.80).

The theoretical values predicted by the average case framework presented in this paper allow several conclusions to be drawn about three algorithms. First, wholist converges to 100% accuracy more quickly than chaining when there is a larger proportion of performance examples, while chaining

$$\sum_{i_0}^{N}\sum_{i_1}^{N-i_0}\binom{N}{i_0,i_1,N-(i_0+i_1)}P_{AC}^{i_0}P_{AB}^{i_1}P_{BC}^{N-(i_0+i_1)} * \prod_{j=1}^{I} nm_{IOSC}(f_j,i_0,i_1,N-(i_0+i_1)) \quad [10]$$

converges more quickly than wholist when there is a larger proportion of foundational examples. IOSC-TM always achieves an accuracy greater than or equal to the accuracy of chaining or wholist. Of course, similar generalizations about the behavior of these algorithms can be drawn by inspecting the algorithms. However, the average-case learning model is able to quantify the exact conditions under which one algorithm will produce more accurate results than another.

3 Conclusion

In this paper, we have presented a framework for average case analysis of machine learning algorithms. Applying the framework consists of 1) understanding how an algorithm revises a hypothesis for a concept, 2) calculating the probability that a training example will be encountered that causes an inaccurate hypothesis to be revised and 3) calculating the effect that revising a hypothesis on the accuracy of the hypothesis. The framework requires much more information about the training examples than the PAC learning model. The information required by the model is exactly the information required to generate artificial data to test learning algorithms. We have applied the framework to three different learning algorithms. We have verified through experimentation that the equations accurately predict the expected accuracy. Although we have currently analyzed only algorithms for conjunctive concepts, we anticipate that the framework will scale to similar algorithms using more complex hypotheses. Our future plans include modeling more complex learning algorithms with more expressive concepts (e.g., *k*-CNF and *k*-DNF) and using statistical techniques to derive the information needed for average case analysis from existing databases.

Acknowledgements

We would like to thank Dennis Kibler, Dennis Volper, Kamal Ali, Nick Flann, Hyam Hirsh and Jeff Shrager for helpful comments on this work. This research is supported by a National Science Foundation Grant IRI-8908260 and by the University of California, Irvine through an allocation of computer time.

References

Atkinson, R., Bower, G. & Crothers, E. (1965). *An introduction to mathematical learning theory.* New York: John Wiley & Sons.

Bruner, J. S., Goodnow, J. J., & Austin, G. A. (1956). *A study of thinking.* New York: Wiley.

Dietterich, T. (1989). Limitations on inductive learning. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 124-128). Ithaca, NY: Morgan Kaufman.

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

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

Flann, N. & Dietterich, T. (1989). A study of explanation-based methods for inductive learning, *Machine Learning.*

Haussler, D. (1987). Bias, version spaces and Valiant's learning framework. *Proceedings of the Fourth International Workshop on Machine Learning,* Irvine CA.

Hirsh, H. (1989). Combining empirical and analytical learning with version spaces. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 29-33). Ithaca, NY: Morgan Kaufman.

Langley, P. (1989). Toward a unified science of machine learning. *Machine Learning,* 3(4).

Mitchell, T. (1982). Generalization as search. *Artificial Intelligence, 18.*

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

Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. *Proceedings of the Seventh National Conference on Artificial Intelligence* (pp. 564-569). St. Paul, MN: Morgan Kaufman.

Pazzani, M. (1990). *Creating a memory of causal relationships: An integration of empirical and explanation-based learning methods.* Lawrence Erlbaum Associates: Hillsdale, NJ.

Restle, F. (1959). A survey and classification of learning models. In R. Bush & W. Estes (Eds.) *Studies in mathematical learning theory.* Stanford: Stanford University Press, pp 415-428.

Sarrett, W. & Pazzani, M (1989a). One-Sided algorithms for integrating empirical and explanation-based learning *Proceedings of the Sixth International Workshop on Machine Learning,* Ithaca, NY: Morgan Kaufman, pp. 26--28.

Sarrett, W. & Pazzani, M (1989b). *Average case analysis of empirical and explanation-based learning algorithms* Technical Report, University of California, Irvine.

Valiant, L. (1984). A theory of the learnable. *Communications of the Association of Computing Machinery, 27,* 1134-1142.

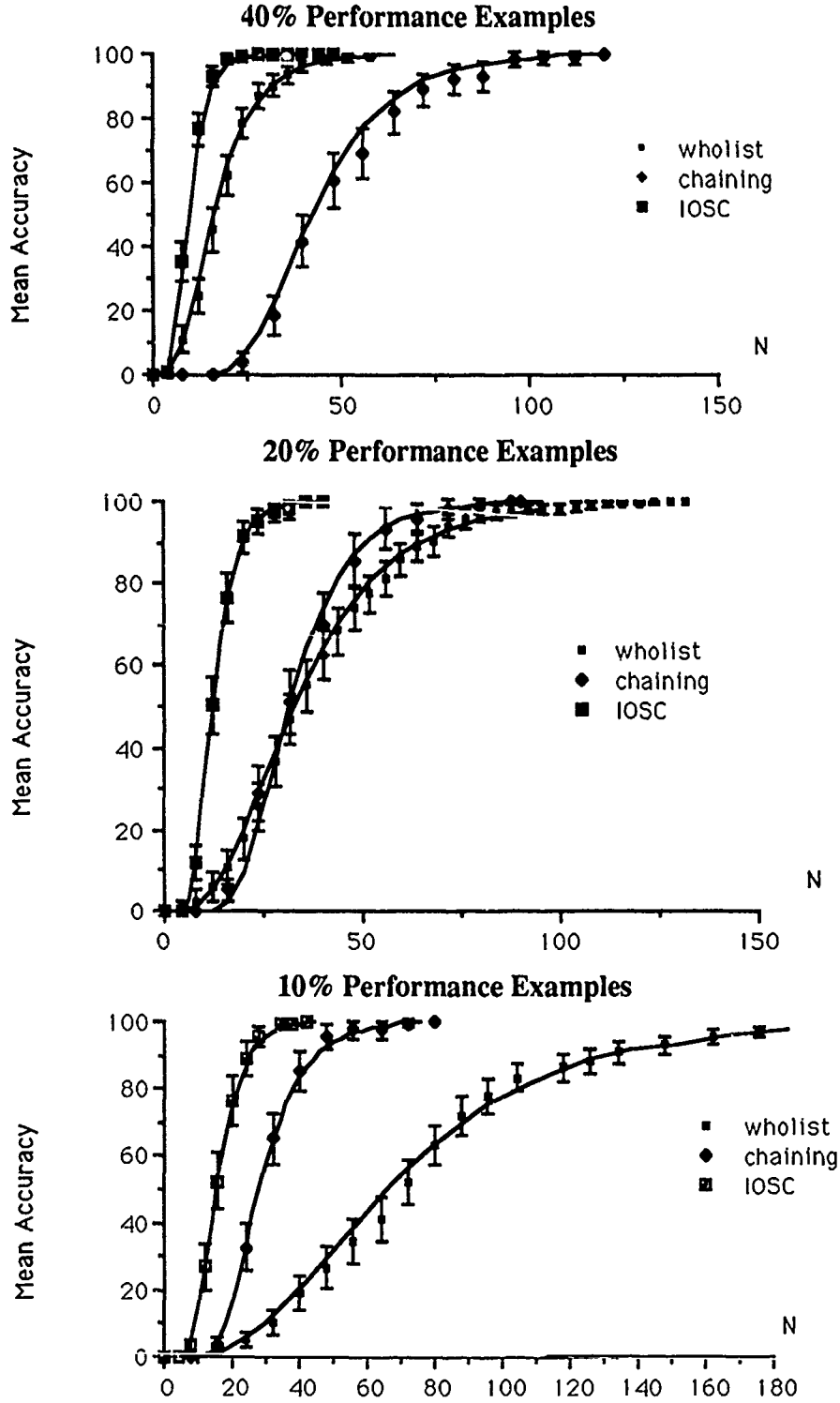


Figure 2: A comparison of the expected and actual accuracy of the three algorithm. The points are the empirical means with 95% confidence intervals and the curves are the value predicted by Equations 3, 7 and 10.

ILS: A Framework for Multi-Paradigmatic Learning

Bernard Silver
William Frawley
Glenn Iba
John Vittal
Kelly Bradford

GTE Laboratories Incorporated
40 Sylvan Road
Waltham MA 02254
USA

Abstract

This paper describes our initial implementation of a domain-independent *Integrated Learning System* (ILS), and one application, which, through its own experience, discovers how to control a telecommunications network. ILS provides a framework for integrating several heterogeneous learning agents, in this case implementations of inductive, search-based and knowledge-based learning. These agents, written in various languages and executing on various platforms, cooperate to improve problem-solving performance. ILS also includes a central controller, called *The Learning Coordinator* (TLC), which manages control flow and communication between the agents using a high-level communication protocol. The agents provide TLC with expert advice. TLC chooses which suggestion to adopt and performs the appropriate actions. At intervals, the agents can inspect the results of the TLC's actions and use this feedback to learn, improving the value of their future advice. At present ILS is being extensively tested, and the initial results are promising.

1. Introduction

This paper describes our initial implementation of a domain-independent, distributed Integrated Learning System (ILS). The first application of ILS determines, through experience, how to control a telecommunications network. This ongoing work addresses issues involved in combining various learning paradigms, integrating different reasoning techniques, and coordinating distributed cooperating problem-solvers.

A wide variety of learning algorithms have been reported in the literature. However, no one algorithm is adequate for a wide range of problems. Our approach is to integrate several algorithms.

ILS provides a framework for integrating several heterogeneous learning agents, written in various languages and executing on various platforms, that cooperate to improve problem-solving performance. It also includes a central controller called *The Learning Coordinator* (TLC) which manages control flow and communication among the agents, using a high-level communication protocol. The agents provide TLC with expert advice concerning the current problem; TLC then chooses which suggestion to adopt, and performs the appropriate actions. The agents compete by offering potentially differing advice to TLC, and cooperate to overcome gaps in their individual knowledge. At intervals, the agents can inspect the results of the TLC's actions and use this feedback to learn. As they learn, either autonomously or cooperatively, the quality of advice given to TLC increases, leading to better performance.

At present, ILS contains three learning agents, NETMAN, FBI and MACLEARN, and a domain simulator called NETSIM. Figure 1 shows the current ILS architecture.

The agents are heterogeneous in that they each have a different:

- model of the domain
- area of expertise in the domain
- learning paradigm

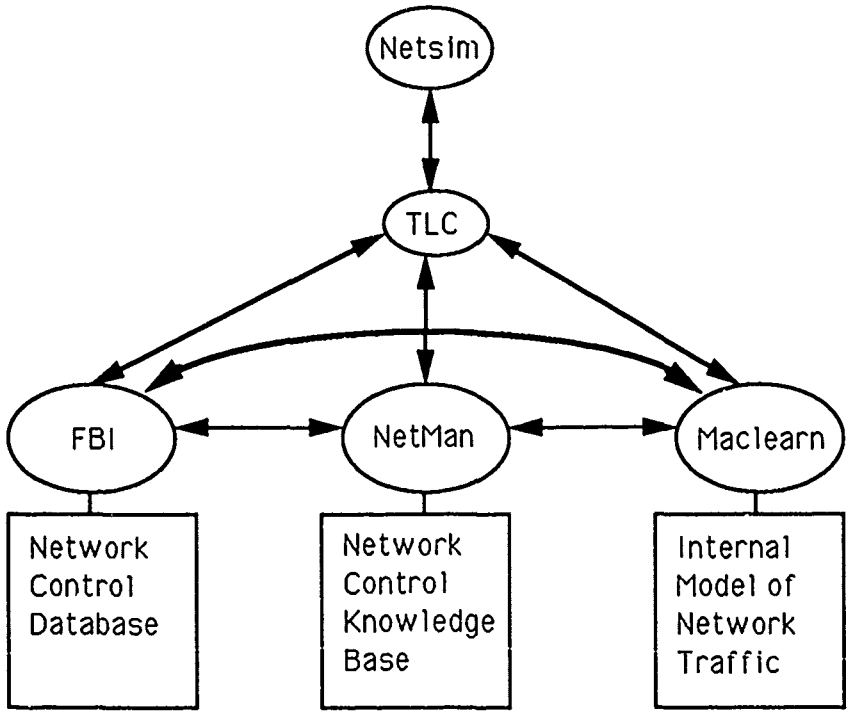


Figure 1: The Integrated Learning System

In addition, TLC, each agent, and the domain simulator can execute in parallel on different machines.

The heterogeneous approach distinguishes ILS from Soar [Laird *et al* 87], which uses one learning paradigm (chunking) for all learning tasks, and THEO [Blythe & Mitchell 89], which uses one representation framework (frames) for all learning tasks. Unlike the system described in [Falkenhainer & Rajamoney 88], ILS can integrate learning agents systems of many different types.

Section 2 briefly describes the learning paradigms and their specific realizations as agents in ILS. Section 3 discusses the domain of experimentation, telecommunications network traffic control, and a realistic simulator called NETSIM. Section 4 describes the ILS communication protocols. Section 5 discusses the operation of TLC. Section 6 describes the inter-agent cooperation implemented in the present version of ILS. Section 7 discusses the possible future directions of this work.

2. Some Learning Agents

This section briefly describes three learning paradigms and their implementations in ILS: inductive (FBI), search-based (MACLEARN) and knowledge-based (NETMAN). FBI and MACLEARN are written in Symbolics Lisp and run on Symbolics Lisp Machines. NETMAN is written in Quintus Prolog and runs on Sun Microsystems workstations.

FBI, MACLEARN and NETMAN are discussed in more detail in [Frawley 89, Iba 89, Silver 90] respectively.

2.1. Inductive Learning

FBI (*F*unction-*B*ased *I*nduction), [Frawley 89], is an extension of Quinlan's ID3, [Quinlan 86]. FBI learns decision trees from large numbers of examples.

Its inputs include:

- the function, f, to be approximated,
- the set of approximators, G,
- the domain, D, (a database) on which f and the members of G are defined,
- and the measure of uncertainty whose local minima are used to define the tree. The default for this function is the *conditional entropy*.

f is a finite-range function which corresponds to the goal concept to be learned. The individual approximators in G are functions which combine and generalize the attributes of the database D.

FBI returns a tree-structured object along with code to evaluate the tree function. As a result of a two-pass procedure, the structure contains no isomorphic subtrees. Thus, branches of a node may represent subsets of values rather than individual values. Each leaf of the object encodes the associated value of f and retains pointers to

all the database entries used in its construction. Each non-leaf node retains pointers to its *inconsistent* database entries, computed as follows. First, during tree construction, classes of inconsistencies are identified and for each class those having the most-likely f-value on the class are used to construct the tree. Then, the inconsistent data that was not used is passed through the nodes and branches down to those non-leaf nodes having no suitable branch, where they are recorded as inconsistent.

In addition, by adding a meta-level field to the database which records all trees computed over it, it is possible to update the trees as the database changes. Each new database entry is passed through the structure of each tree. If it is consistent with a tree's approximation to f it is indexed by some leaf node of the tree; otherwise, it is indexed by the inconsistencies of some non-leaf node.

Unlike ID3 which is used to build one decision tree, FBI is used to build and maintain a sequence of trees employing different approximators or different methods of approximation.

- Function-based induction can iteratively refine an approximation to a particular classifier in this way: First, a decision tree based on attributes alone is computed. Then, the tree is examined by the user for intelligibility or domain or statistical relevance. Some attributes may be removed from the set of approximators; new domain and context functions may be added to the approximators. Then the tree is recomputed, re-examined, etc.

- Consultation programs which utilize different rulesets can be built: Various aspects of domain or context knowledge may be suggested by different experts or domain models. These can be encoded into different decision trees and the statistics of each and performance over time compared.

- Computed decision-tree functions can be used as approximators in constructing other decision-tree functions.

There are two ways in which FBI makes use of domain and contextual knowledge in constructing approximations. First, *the approximator functions can appropriately combine attributes.* For example, in network traffic control, counts of call attempts and call completions are recorded for all trunk groups for each five-minute period; but these numerical attributes are not individually important. What is important is the classification of ratios of completions to attempts to indicate over- or under-utilization of given trunk groups. The use of thresholds for certain ratios of attributes exemplifies how domain knowledge directs attribute combination; the situation-dependent values of thresholds used exemplify contextual knowledge. Second, *the selection of the next best-approximator can depend on domain or context.* In straightforward statistical induction, a decision tree is constructed recursively by selecting, at each level, the approximator which minimizes an uncertainty function, u, over the unresolved data. By allowing the user to specify the selection method, FBI can override strict minimization criteria. For example, domain knowledge might dictate that the approximator g should appear above the approximator h in any tree in which h occurs. Or contextual information unrelated to the uncertainty calculation may provide weights $\{a_i\}$ adjusting u to select the minimizer of $a_i g_i$.

FBI also includes a function-discovery mode which discovers two types of *interesting* functions. First, there are classifiers defined by tree nodes which, during tree construction, have replicated immediate subtrees. Secondly, disjunctive concepts representing paths to maximal subtrees replicated at different levels in the tree are automatically defined. This extends the ongoing work in the field on learning disjunctive normal form (DNF) concepts using decision trees as a concept description language, [Pagallo & Haussler 89, Pagallo 89, Matheus & Rendall 89].

Often the concepts discovered in this way are useful because they reflect genuine features of the domain. In other cases, the concepts discovered are artifacts caused by random patterns in the example set. FBI can ask other agents of ILS, in particular NETMAN, to assess the utility of a discovered concept. This is discussed in Section 6.3.

2.2. Search-based Learning

MACLEARN [Iba 89, Iba 88] currently performs *best-first search* (see, for example, [Nilsson 80]) in order to learn *macro-operators*, or *macros*, which are useful combinations of operators that can be subsequently treated as a single operator. Macro-learning is a form of chunking that can improve search performance by enlarging the set of operators available for the search. The availability of a good set of macros will often drastically reduce the combinatorially explosive nature of a search problem.

Macros reduce search in two general ways. The first is to take larger steps in the search space. Since a macro actually represents a number of primitive operators, the application of a single macro can result in moving a greater distance through the search space. A problem that may require hundreds of primitive steps to solve, may be solvable more quickly by only tens of applications of macro steps.

The second way is to provide synergistic combinations of operators: applying a set of operators as a group may be beneficial, whereas applying any operator alone may make things worse. In such a case, it may be difficult for a search to discover the combination; but once it has been discovered it is valuable to remember it in order to avoid having to duplicate the lengthy search in subsequent similar situations. Thus macro learning can be viewed as a kind of encapsulation of experience.

However, there is a cost associated with creating macros. Macros enable larger steps through the search space, but they increase the number of possible branches at each step. This increased branching factor will tend to slow down the search. In order for macros to prove beneficial, the advantages must more than compensate for the disadvantages. Thus a macro learning program must have a means of deciding which macros to keep and which to discard. MACLEARN uses various heuristic criteria to perform this filtering process.

On complex problems, MACLEARN may encounter a combinatorial explosion as the search space of possible operators becomes too large. In such cases, MACLEARN becomes bogged down in the search and may be unable to find a satisfactory solution. Other agents within ILS can provide assistance to MACLEARN by constraining the search and indicating which part of the search space should be examined. This is discussed further in Section 6.2.

2.3. Knowledge-Based Learning

NETMAN, [Silver 90], is an example of a knowledge-intensive learning system. The algorithm used is based on *explanation-based learning* (EBL), [DeJong & Mooney 86, Mitchell *et al* 86], modified to work with an imperfect domain theory, [Silver 86, Silver 88].

NETMAN starts with a large amount of domain knowledge, expressed as rules. However, the knowledge need be neither complete nor totally accurate. As a result, NETMAN can make mistakes. (Human experts suffer from the same limitation, of course.) In EBL terms, the domain theory is incomplete and computationally intractable, and so NETMAN uses a heuristic approximation.

NETMAN learns four major types of information from experience:

1. *Stored caches*: NETMAN stores as a macro the sequence of rule firings that led to advice that worked.
2. *Support List*: The support list indicates how successful or unsuccessful a particular action proved to be. Those that have proved valuable in the past are more likely to be used in the future.
3. *Possible Bugs*: When an action fails to achieve the expected effect, NETMAN can classify the cause and severity of this failure. This information is stored and will affect future use of the action.
4. *Plans*: A plan consists of a sequence of actions that have proved useful, together with the expected effect of each action.

When actions have unexpected effects, NETMAN attempts to explain the cause of the unpredicted behavior. This analysis allows NETMAN to discover that sometimes an action will fail due to a *bug*. NETMAN is able to classify the type of bug, and associate this type with the action. Note that the action may often work successfully; bugs may occur only in certain situations.

Ideally, NETMAN would be able to accurately distinguish between situations in which an action will be successful and those in which it will fail with a bug. Unfortunately, the computation involved, and the stochastic nature of the domain, make this impossible to do precisely. Instead, NETMAN heuristically differentiates the cases by calling on another component of ILS, FBI. Section 6.1 describes this interaction.

3. The Problem-Solving and Learning Tasks

Consider a system or process characterized by a time-dependent internal state, S, whose quality or merit at any given time is described by an evaluation function e with values in the interval [0,1]. The system responds with a time lag to service demands and to whatever *control* has been imposed on it. The system goal is to operate with its evaluation at or near 1. In complex systems, subject to widely varying demand and the possibility of subsystem failure, this can be difficult. At certain discrete times, t_i, an external controller imposes one of N controls, $A_1, \ldots, A_N$, in order to increase the value of $e(t)$ at subsequent times. One way to measure the effectiveness of the choice of a control at time t_i is to compare the states just before and sometime after the control action. For example, with $e=e(S(t_i))$ and $e'=e(S(t_i+\Delta))$, a simple function such as $f(e,e')=(e'-e)/(1-e)$ may suffice. The number of control options, N, is one measure of the complexity of system control.

The problem-solving role of a controlling agent is to observe $e(S(t))$ or $S(t)$ over time and, at times of its own choosing, to actuate new choices of control. In the absence of an adequate model of the system, it may not be possible to assess how well a problem-solver is doing. But, by presenting the same or similar situations to two competing controllers, it is possible to determine which is more effective. The learning role of a controlling agent is to improve based on experience; that is, the self-adapted agent is to be more effective than the agent in its original state.

3.1. Application Domain: Network Traffic Control

The dynamic system control problem used as the basis for ILS experimentation is the control of traffic in a circuit-switched telephone network. Network state is defined by a considerable volume of data regarding call placement and switch and trunk-group usage provided on a minute-by-minute basis. The overall network evaluation function used here is the percentage of attempted calls that are successfully completed, averaged over the most recent five-minute interval. Generally, human network traffic managers strive to keep this value above 99% but there are many types of situations in which a considerable percentage of calls fail and intervention is required.

The task is difficult, partly because of the huge amount of data produced and its time-varying nature. There is some reliance on *preplans*; standard procedures for predictable occurrences, such as Mother's Day, which is the busiest calling day of the year, or for special contests that radio stations occasionally offer ("We will give two free tickets for tomorrow's show to the first ten callers with the correct answer..."). However, unpredictable problems arise, and they cause the major difficulty. One example is the partial or total failure of a network element (a trunk group or a switch) which may cause some unavoidable denial of service to some users; however, effective traffic management can greatly improve the situation, for example, by rerouting traffic around the failed element.

Many other Artificial Intelligence techniques have been applied to Network Traffic Management, including Case-Based Reasoning [Kopeikina *et al* 88], Distributed AI [Adler *et al* 89, Brandau & Weihmayer 89] and traditional Expert Systems (*e.g.* [Kosieniak *et al* 88]). These approaches suffer from the inflexibility of all non-learning programs: an inability to learn from experience and thereby to improve their control of the network.

3.2. The Simulator

The performance module for ILS experiments is a network simulator called NETSIM, [Frawley *et al* 88]. NETSIM conducts a fine-grained simulation of the call placement process in a network of end-offices and tandem switches and implements a set of controls applicable to switches and trunk groups. One property of this domain simulation is its complexity. In ongoing experiments on a ten-switch, sixteen-trunk network there are always at least 3000 individual "legal moves". Moreover, in many cases it is reasonable to impose several controls simultaneously, increasing the number of allowable control options considerably.

4. The ILS Protocol

ILS is completely distributed. Within ILS, agents interact via TCP/IP streams using a communication protocol consisting of the six message (request) types enumerated below. Any agent may access any other agent using ILS language primitives or any other calls understood by the target agent.

1. The INITIALIZE message is a request for an agent to initialize itself and set up communication with other agents. This message includes information specifying the current host machine for each of the other agents. The response to this message is simply an acknowledgment that system and communication initialization is complete.

2. The ADVISE message is a request for advice on what to do in the current state of the external domain to be controlled. An agent responds to this message by examining the current state and deciding what actions would be best for improving this state. A vote (in the range 1-5) accompanies each recommendation, indicating how beneficial the agent predicts this advice is likely to be. If the agent is unable to come up with advice that improves the situation, it may respond with the reply UNKNOWN.

3. The CRITIQUE message is a request for an agent to provide its opinion of advice offered by other agents. The specific pieces of advice other agents have proposed are passed along as part of this message. The response to this request is a set of votes (again in the range 1-5) reflecting the value of each piece of advice as viewed by the responding agent. A vote of UNKNOWN is given for those pieces of advice that an agent is not able to meaningfully critique.

4. The DATA-AVAILABLE message is used to notify agents which of the last suggested sets of actions were taken and that a sufficient period of time has passed since they were taken to allow domain data to reflect those actions. In response to this message, agents can gather feedback from the domain in order to learn.[1] Finally, they send back an acknowledgment that they have finished processing the current data.

5. The RECORD message is sent to agents which maintain their own databases of past cases. One agent asks another to record additional information about a given case for later recall or processing. For example, this primitive is used by NETMAN to select special cases for processing by FBI. If the message recipient does not have a recording facility, it returns UNKNOWN.

6. The CLASSIFY message is a request for an agent to classify the current state of the external domain with regard to a particular predicate used in a previous RECORD message. For example, NETMAN uses this message to ask FBI to classify the current state using a tree constructed from states that have been RECORDed.

[1] The technique used to gather data from the domain is agent-specific.

5. The Learning Coordinator

In its current state, the ILS relies on a Learning Coordinator (TLC) to manage the control flow between the agents and the simulated domain. The basic loop, repeated every five simulated minutes, is as follows:

1. TLC asks the agents to propose actions to control the network. Each agent returns a list of possible actions, and associates with each action a *vote*, a number between 1 and 5, indicating the agent's perception of the value of that action.

2. TLC then asks the agents to critique the proposals of the other agents. Each agent returns a vote for each proposal.

3. Now TLC has a list of proposed actions, and each action has, at present, a set of up to three votes associated with it. TLC has to choose between these proposals. There are several possible techniques that can be used for this selection process, some of which are described below. As currently implemented, TLC averages the votes for each proposal and executes the action with the highest average score.

4. Five simulated minutes later new switch statistics are produced. The agents inspect these statistics to obtain feedback on the success or failure of the chosen action, allowing them to learn appropriately to affect their future problem-solving performance.

The following sections discuss the individual steps in more detail, using NETSIM as the domain simulator.

5.1. Proposing Controls

The agents individually obtain network statistics directly from NETSIM via TCP/IP streams. Each agent examines the statistics and attempts to produce an ordered list of actions that the agent believes will improve the network state. An action may remove some controls already in place and/or impose some new controls.

Alternatively, an agent can suggest taking no action, if it believes that the current state of the network is satisfactory, or it can indicate that it does not know what to do. In the last case, no further processing is performed.

No agent will propose actions that appear to make the situation worse. If an agent cannot find an action that will improve a bad network state, it will indicate that it does not know what to do.

Each agent must also calculate a *vote* for each proposed action; in other words, determine how much it believes in the action that it is proposing. In the current implementation, each agent calculates the vote in a different way, described below. The architecture does not require that the agents have a uniform semantics for voting. However, each agent should be internally consistent; an agent should expect that actions it scores highly will perform better than those actions to which it gives a lower score.

MACLEARN calculates the vote using an internal simplified simulator, called FLOSIM. It compares the simulated outcome with the current network state, giving a very high vote if the action appears to greatly improve the situation, and a somewhat lower vote if the action promises only a small improvement.

FBI calculates its vote using its database of previous examples. If the proposed action has greatly helped similar situations in the past, it receives a very high vote, otherwise a somewhat lower vote. An example consists of descriptions of an initial network state, a set of actions, and the network state that results from taking those actions and running the network for an additional period of simulated time. Two preferred subsets are maintained, one of examples whose rankings are *very high*, and one for examples whose rankings are *high*. Learned decision trees defined over these subsets map an initial state to a set of actions: the advice provided is either a *very high* set of actions, a *high* set, or none. FBI's vote depends on success ratios of similar actions in the database.

NETMAN bases its vote partly on the basis of past experience. It also uses its domain theory, taking into account the effect of controls. For example, an action receives credit if it contains controls that move traffic from overflowing trunks to non-overflowing ones. NETMAN calculates the vote for a proposal by combining the vote based on past experience (if available) with the vote based on domain knowledge.

5.2. Critiquing Suggestions of Others

In general, an agent critiques a proposal using the same mechanism used to produce a vote. Thus MACLEARN uses FLOSIM to simulate the effect of the proposed controls, FBI uses a decision tree, computed over all examples, mapping pairs of initial states and actions into rankings to evaluate the expected effect of a proposed action, and NETMAN uses a domain model and previous experience to estimate the effectiveness of the proposal. NETMAN also performs a series of "sanity checks" on the proposal. For example, rerouting traffic to a trunk that is currently non-operational is a bad mistake, and a proposal containing such a reroute will receive a poor score.

In some cases an agent may indicate that it has no opinion on a proposal. This can happen if the agent has no knowledge or experience concerning a particular control or set of controls. Such critiques are discarded.

5.3. Choosing between the Proposals

As indicated above, currently ILS chooses the action with the highest average score. Other alternatives include:

- Choose the action with the highest *raw* score; in other words ignore the effects of critiquing.

- Choose the action that is highly scored by more than half of the agents.
- Choose the action proposed by the agent that has proved most reliable in the past.
- Weight the votes of an agent in proportion to its past performance.

These strategies are currently being evaluated.

The last two suggestions above require that TLC keep track of the performance of each agent, and thus itself *learn* to choose among the various proposals. This can be done with varying degrees of sophistication. The simplest method is to increase the perceived reliability of an agent when its choice is selected and causes improvement, and to decrease the perceived reliability, if its action makes the network worse.

One problem with this approach is that TLC cannot know whether a successful action was the best of all of the suggested actions; this is inherent in every scheme that does not perform a total search. Thus an agent may be rewarded when the untried suggestion of another agent would have caused greater improvement. Another difficulty with this method is that the agents are learning, so that an agent that originally provided poor advice may now produce effective suggestions. One way of correcting this problem is to discount earlier performance, giving more weight to recent experience.

Nevertheless, empirically the simple technique appears promising. It can be extended to the critiquing process as well; if an agent gave a low score to an action that was in fact successful, TLC can decrease its perceived reliability and similarly in the other cases.

All of the techniques described above result in TLC selecting one of the proposed actions. A more sophisticated approach would be to form new advice by combining the proposals of individual agents. Various combination techniques could be used. For example, a purely syntactic method might use union or intersection operations to generate the new advice. Alternatively, a more knowledge-based approach might detect that two of the proposals treat independent separable problems, and so could be usefully combined.

5.4. Obtaining Feedback

One interesting feature of the ILS architecture is that some agents can learn even when another agent's action is the one that is chosen. In the case of FBI, the mechanism is particularly simple; the whole cycle is just another example, consisting of a before state, an action and an after state. NETMAN is able to learn if the action chosen is sufficiently similar (according to a complex metric) to an action it proposed.

At present, MACLEARN does not make use of the feedback.

6. Inter-agent Cooperation

Section 5.2 described one level of cooperation between the agents: the ability to critique the actions of each other. This section further describes our ongoing work on some of the ways in which the various agents interact to improve the performance of individual agents. In the current system, *only the first mechanism is implemented*; the others will be added shortly.

6.1. Concept Formation

Ideally, NETMAN would be able to distinguish accurately between situations in which an action will be successful and those in which it fails in a certain way. However, the complexity and stochastic nature of the domain make this an unrealistic target. Instead, NETMAN heuristically differentiates the cases by calling on FBI.

FBI is given two classes of examples: in one class are all the examples where the failure mode occurred, in the other are the cases where the action worked. FBI is used to produce a function that heuristically classifies network states as likely to suffer from that failure mode or not. If NETMAN is considering performing the action, it asks the inductive learning program for its classification. If the current state is classified as likely to fail, and the failure mode has proved sufficiently severe, NETMAN does not propose the action. In effect, the classification step is added as a precondition to the treatment rule that proposes the action.

6.2. Reducing the Search Space

As mentioned previously, MACLEARN can be overwhelmed by the combinatorial explosion on complex networks. This problem could alleviated by giving MACLEARN *constrained* search problems. In a future version of ILS, the constraints would come from FBI or NETMAN detecting important features of the current network state. The search could be constrained in various ways:

- Reduced operator sets: MACLEARN is told to consider only certain types of controls.
- Constrained areas of application: MACLEARN is told to consider placing controls on only the specified network elements.
- New starting state: MACLEARN is told to assume that certain controls must be present.

6.3. Assessing Discovered Concepts

As described in Section 2.1, FBI can automatically discover potentially interesting concepts that are combinations of other existing functions. Often the concepts discovered in this way are useful because they reflect genuine features of the domain. In other cases, the concepts discovered are artifacts caused by random patterns in the example set.

A knowledge-based agent, such as NETMAN, could be used to heuristically classify a concept as useful or not

useful. When FBI proposes a new concept, it could ask NETMAN if there is any domain knowledge that combines the functions contained in the concept. If NETMAN did have such domain knowledge, the concept would probably be a genuine one. If not, the concept may be an artifact, although perhaps NETMAN is missing some domain knowledge.

7. Conclusions and Further Work

ILS is a framework for integrating several distributed heterogeneous learning agents that cooperate to improve problem-solving performance. The agents learn both independently and cooperatively. Each agent has been tested independently using telecommunications traffic control as a domain, and each has demonstrated that it can learn by interacting with that domain. At present ILS is being extensively tested using the same domain. One focus of the current research is the relative utility of various TLC selection strategies. Simple strategies appear to be effective.

7.1. Further Work

There are several directions that the research is expected to follow in the near future:

- Detailed experiments will be performed to evaluate the ILS framework. At a minimum, to be successful, ILS must demonstrate:
  - *Enhanced problem-solving performance*: The ILS as a whole must do better than each of its constituent parts.
  - *Improved Learning*: The learning of each agent will be accelerated within ILS, and each agent's knowledge will be adjusted to be more effective.
- More learning agents will be added in order to assess the completeness of the communication protocols and to explore inter-agent cooperation. The next to be added will be a pattern-based learning program such as AQ11 [Michalski & Chilausky 80].
- Other forms of search and methods for constraining them will be examined. This effort will enhance the work described in Sections 2.2 and 6.2 on best-first search and on how agents can provide advice to constrain such searches.
- The areas of inter-agent cooperation will be expanded.
- ILS will also be tested in another, yet to be determined, application domain.

Another possibility under active consideration is to remove TLC by distributing its responsibilities amongst the learning agents. This approach allows the agents to behave more autonomously, and draws on the existing work in Distributed Artificial Intelligence, [Brandau & Weihmayer 89].

The intent is ultimately to develop a *domain independent* ILS.

7.1.1. Acknowledgments

Many other members of GTE Laboratories, including Oliver Selfridge, Alvah Davis, Mark Adler, Alan Lemmon, Ralph Worrest, Rich Brandau and Ludmila Kopeikina provided the project with useful information concerning the network traffic control domain, and have provided interesting discussions in machine learning and associated fields like case-based reasoning. Tom Fawcett built NETSIM and contributed many valuable ideas to the project. June Pierce drew the figure and helped in the preparation of the paper. Comments from Chris Matheus, Marlene Kliman, Rich Sutton, Chuck Anderson and two anonymous referees greatly improved the presentation of the paper.

7.1.2. References

Adler *et al* 89. Adler, M., Davis, A.B., Weihmayer, R. and Worrest, R. Conflict-resolution Strategies for Nonhierarchical Distributed Agents. In Gasser, L. and Huhns, M.N. (editors), *Distributed Artificial Intelligence, Volume 2*, 1989, pp. 139-161.

Blythe & Mitchell 89. Blythe, J. and Mitchell, T.M. On Becoming Reactive. In Segre, A. (editor), *Proceedings of the 6th International Machine Learning Workshop*, 1989, pp. 255-257.

Brandau & Weihmayer 89. Brandau, R. and Weihmayer, R. Heterogeneous Multiagent Cooperative Problem Solving in a Telecommunication Network Management Domain. In Benda, M. (editor), *Proceedings of 9th Workshop on Distributed Artificial Intelligence*, American Association for Artificial Intelligence, 1989, pp. 41-58.

DeJong & Mooney 86. DeJong, G. and Mooney, R. Explanation-Based Learning: An Alternative View. *Machine Learning 1*, 2 (1986), 145-176.

Falkenhainer & Rajamoney 88. Falkenhainer, B. and Rajamoney, S. The Interdependencies of Theory Formation, Revision, and Experimentation. In Laird, J. (editor), *Proceedings of the 5th International Machine Learning Workshop*, 1988, pp. 353-366.

Frawley 89. Frawley, W.J. Using Functions to Encode Domain and Contextual Knowledge in Statistical Induction. In Piatetsky-Shapiro, G. and Frawley, W.J. (editors), *Knowledge Discovery in Databases (IJCAI-89 Workshop)*, International Joint Conference on Artificial Intelligence, 1989, pp. 99-108.

Frawley *et al* 88. Frawley, W.J., Fawcett, T.E. and Bradford, K. *NETSIM: An object-oriented simulation of the operation and control of a circuit-switched network.* Tech. Rept. TN 88-506.1, Computer and Intelligent Systems Laboratory, GTE Laboratories Incorporated, 1988.

Iba 88. Iba, G.A. *The Application of Heuristic Search and Discovery of Macro-operators to Network Traffic Control.* Technical Memorandum TM-0062-10-88-506, Computer and Intelligent Systems Laboratory, GTE Laboratories Incorporated, 1988.

Iba 89. Iba, G.A. A Heuristic Approach to the Discovery of Macro-operators. *Machine Learning 3*, 4 (1989), 285-317.

Kopeikina *et al* 88. Kopeikina, L., Brandau, R. and Lemmon, A. Case Based Reasoning for Continous Control. In Kolodner, J (editor), *Proceedings of a Workshop on Case-Based Reasoning*, DARPA, 1988, pp. 250-259.

Kosieniak *et al* 88. Kosieniak, P., Mathis, V., St Jaques, M. and Stevens, D. The Network Control Assistant (NCA), a Real-time Prototype Expert System for Network Management. In *Proceedings of First International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems*, Association for Computing Machinery, 1988, pp. 367-377.

Laird *et al* 87. Laird, J.E., Newell, A. and Rosenbloom, P.S. Soar: An architecture for general intelligence. *Artificial Intelligence 33*, 1 (1987), 1-64.

Matheus & Rendall 89. Matheus, C.J. and Rendell, L.A. Constructive Induction on Decision Trees. In Sridharan, N.S. (editor), *Proceedings of the Eleventh IJCAI*, International Joint Conference on Artificial Intelligence, 1989, pp. 645-650.

Michalski & Chilausky 80. Michalski, R.S. and Chilausky, R.L. Learning by Being Told and Learning from Examples. *International Journal of Policy Analysis and Information Systems 4*, 2 (1980), 125-161.

Mitchell *et al* 86. Mitchell, T.M., Keller, R.M. and Kedar-Cabelli, S.T. Explanation-Based Generalization: A Unifying View. *Machine Learning 1*, 1 (1986), 47-80.

Nilsson 80. Nilsson, N.J. *Principles of Artificial Intelligence.* Tioga Pub. Co., Palo Alto, California, 1980.

Pagallo 89. Pagallo, G. Learning DNF by Decision Trees. In Sridharan, N.S. (editor), *Proceedings of the Eleventh IJCAI*, International Joint Conference on Artificial Intelligence, 1989, pp. 639-644.

Pagallo & Haussler 89. Pagallo, G. and Haussler, D. Two Algorithms that Learn DNF by Discovering Relevant Features. In Segre, A. (editor), *Proceedings of the 6th International Machine Learning Workshop*, 1989, pp. 119-123.

Quinlan 86. Quinlan, J.R. Induction of Decision Trees. *Machine Learning 1*, 1 (1986), 81-106.

Silver 86. Silver, B. Precondition Analysis: Learning Control Information. In Michalski, R.S., Carbonell, J.G. and Mitchell, T.M. (editors), *Machine Learning: An Artificial Intelligence Approach Vol 2*, 1986, pp. 647-670.

Silver 88. Silver, B. A Hybrid Approach in an Imperfect Domain. In DeJong, G. (editor), *Proceedings of AAAI Symposium on Explanation-Based Learning*, American Association for Artificial Intelligence, 1988.

Silver 90. Silver, B. *NetMan*: A Learning Network Traffic Controller. In Matthews, M. (editor), *Proceedings of Third International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems*, Association for Computing Machinery, 1990.

An Integrated Framework of Inducing Rules From Examples

Wu, Yihua
Wang, Shulin
Zhou, Qing
Institute of Computing Technology
Academia Sinica
P.O. Box 2704–3, Beijing 100080
P.R. CHINA

ABSTRACT

The paper presents an integrated framework, namely IR1, for inducing the inference rules from examples. Achieving this requires considering the generality, inducibility, incrementality, uncertainty and hierarchy of expert rules. Our approach is a mutual integration of empirical and analytical techniques to avoid shortcomings of them when applied individually. It also incorporates some interesting ideas or algorithms to achieve these goals. Primary experimental results are encouraging and more work is required.

1. Introduction

Quinlan's(1983,1986) ID3 is an interesting algorithm which induces a decision tree from examples. Each example is depicted by several attributes and belongs to a class. The task is to derive a logical description of each class. The significance of the problem varies with the different appearances of the description. For instance, if each class represents a decision and each attribute an aspect of the situation, the description may be interpreted as a decision rule; if each class corresponds to a kind of disease and attribute a symptom, the description may be interpreted as a diagnosis rule. In other words, solving the problem could greatly contribute to widening the knowledge acquisition bottleneck by inducing rules automatically.

A lot of improvements have been done since ID3. For instance, Schlimmer & Fisher (1986) constructed ID4---an incremental ID3; Utgoff(1988) built ID5---a more elaborate incremental ID3; Cendrowska(1987)'s PRISM is able to obtain a set of the simple classification rules instead of a decision tree; Quinlan (1987) also simplifies decision tree into a set of modular production rules. However, the rules or trees induced by them hardly capture the characteristics of expert rules used in making inferences. The paper is totally devoted to solving this problem. We begin our approach by examining characteristics of human inference rules. A learning framework is then presented to integrate similarity–based and explanation–based approaches to induce rules.

2. Exemplary Domain

We give an exemplary problem excerpted from Cendrowska(1987). An adult spectacle wearer wants to purchase her first pair of contact lenses. In optician's point of view, this is a three–category classification problem. His decision will be one of:

@1: the patient should be fitted with hard contact lenses,
@2: the patient should be fitted with soft contact lenses,
@3: the patient should not be fitted with contact lenses.

In reaching his decision he must consider one or more of four factors:

a: the age of the patient
1. young, 2. pre–presbyopic, 3. presbyopic
b: her spectacle prescription
1. myope, 2. hypermetrope
c: whether she is astigmatic
1. no, 2. yes
d: her tear production rate
1. reduced, 2. normal

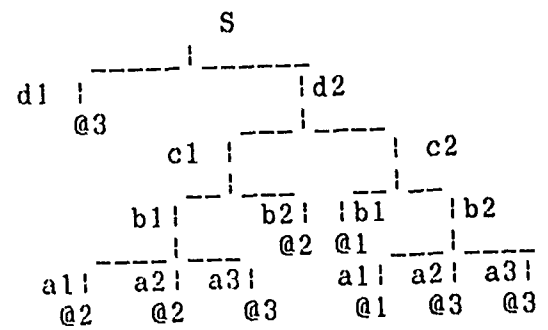
Table 1 shows the optician's decision for each combination of the four factors. However, the optician just uses his rule–based experience instead of carrying such a table around with him, either on his person or in his head.

TABLE 1. decision table for fitting contact lens

Value of attribute a b c d	decision @	Value of attribute a b c d	decision @
1 1 1 1	3	1 2 2 1	3
1 1 1 2	2	1 2 2 1	1
1 1 2 1	3	2 1 1 1	3
1 1 2 2	1	2 1 1 2	2
1 2 1 1	3	2 1 2 1	3
1 2 1 2	2	2 1 2 2	1
2 2 1 1	3	3 1 2 1	3
2 2 1 2	2	3 1 2 2	1
2 2 2 1	3	3 2 1 1	3
2 2 2 2	3	3 2 1 2	2
3 1 1 1	3	3 2 2 1	3
3 1 1 2	3	3 2 2 2	3

Based on Table 1 as the training set, ID3 will obtain a decision tree and PRISM will produce a set of rules(Fig.1). The obvious improvement by PRISM is that its rules are similar to the ones the optician may use. Its fundamental improvement lies on its eliminating redundance. For example, if the patient is a presbyope with high hypermetropia and astigmatism(i.e., a3&b2&c2), the optician would know immediately that she was not suitable for contact lens wear. PRISM can reach this conclusion by using rule 9. However, ID3 needs to know her tear production (value of d) in order to make decision. This test is normally carried out with a lot of time and fee. It would be quite understandable if the patient becomes upset or angry on finding the test is unnecessary. The consequence could be more serious if the attribute involves surgery.

ID3's decision tree:



PRISM's rules:

1. c2 & d2 & b1 = = > @1
2. a1 & c2 & d2 = = > @1
3. c1 & d2 & b2 = = > @2
4. c1 & d2 & a1 = = > @2
5. c1 & d2 & a2 = = > @2
6. d1 = = > @3
7. a3 & b1 & c1 = = > @3
8. b2 & c2 & a2 = = > @3
9. b2 & c2 & a3 = = > @3

Fig. 1. The output of ID3 and PRISM

3. Characteristics of inference rules

In spite of such an improvement, PRISM, in its essence, is a classification system and requires providing all the necessary attributes prior to decision making. We still find the differences between PRISM's rules and expert rules(or inference rules). These include:

A. Hierarchy. The inference rules are being linked together to form an inference link. The intermediate conclusion may be a suggestion of having a tear production test, when the patient is hypermetrope and astigmatic. In other words, human expert does not have all necessary information ahead of decision making. He must reason what should be done when meeting with incomplete information. This is one of important reasons for rule hierarchy. Another reason is that there exist some important intermediate concepts or decisions. This also contributes to forming an inference link in order to reach the final conclusion.

B. Theory Dependency. Rules may be derived

from a single example with aid of domain knowledge. Theory plays a central role in justifying derived rules.

C. Incrementality. The human knowledge is accumulative. When new experience(an example in our case) comes up, human expert can refine the old knowledge base. The problem in our case is how to modify the generated rule base to include the random new examples.

D. Generality. An extensive generalization is done in deriving rules. One essential generalization is the introduction of the variables from objects. This explains the predictive power of human knowledge.

E. Uncertainty. Uncertainty is typical of human knowledge. People frequently do not know if an example belongs to a class surely. Instead, they prefer to make assertions in a plausible way.

4. An Integrated Approach: IR1.

In order to include the above improvements, we create a new approach of integrating similarity–based(SBL) and explanation–based methods(EBL), namely, IR1.

4.1. Integration Issues

Integrating SBL and EBL, or preferably empirical and analytical learning, is considered to be feasible to avoid shortcomings of them when applied individually. A lot of researchers are working on different integration models(Lebowitz, 1986; Danyluk, 1987; Kedar–Cabelli, 1987; Pazzani, 1988). A striking similarity shared by these models is that one form of learning is invoked first and the results are integrated by the other. This one–way interaction is not the way man acquires knowledge and is not expected to be efficient(Swaminathan, 1989). This fallacy is studied and tackled by Swaminathan(1989). However, his model works on concept formation. New points must be considered when trying to derive inference rules usable in KBSs. Let us go into details of IR1(see Fig. 2).

IR1 has two entries which correspond to two running versions: incremental or in batch. Initially, no rules but a large set of examples exist and IR1 will be invoked in batch to trigger empirical components. A set of inference rules will be derived and the system is ready to explain new examples. Each time a new example gets in, IR1 will invoke its analytical component EBL to explain the example. When explanation result is inconsistent, incorrect or incomplete, empirical components will be invoked incrementally to update the rules generated before. The update is guided by making full use of explanation results which determine which rules are to be modified. This procedure will be repeated until a satisfactory explanation is achieved or the new example is found contradict with an old one. As a result, the updated rules are able to cover new example as well as old ones. At this moment we see that incrementality is achieved by adopting EBL techniques, and also that a mutual integration is done by an iterative interaction between empirical and analytical components.

BATCH: Generating classification rules with CLASS;
Establishing a primary rule hierarchy with EPH;
Generalizing rules with GR;
Refining hierarchy with ERH;
INCREMENTAL: Explaining new examples with analytical component EBL;
Triggering empirical components toupdate rules in terms of explanationresults as bellow;
switch (result.type)
{ case perfect–explanation:
outputing successful explanation information;
making explanation–based generalization with EBG;
break;
case incorrect–explanation:
modifying rules and hierarchy with Incremental CLASS and EPH;
regeneralizing rules with Incremental GR;
Refining hierarchy with Incremental ERH;
break;
case inconsistent–explanation:
same as above but with different arguments;
case incomplete–explanation:
regeneralizing rules with Incremental GR;
refining hierarchy with Incremental ERH;
break;
case counter–example:
outputing the old example inconsistent with the new one;
break;
}
repeating above explanation and modification until a successful explanation or a counter–example;

Fig.2 IR1's flowchart in C–like language

4.2. Representation Issues

An example in IR1 is a conjunctive combination of assertions about objects. The assertion is the most primitive semantic unit and can be generally represented as:

(OBJECT, ATTRIBUTE, VALUE, CERTAINTY),

where OBJECT is a specific instance of a concept. For example, "John is surely 24 years old" is represented as:

(John, Age, 24, surely),

where John is a specific instance of man. This representation makes it possible to form variables from a set of objects. However, this basic representation is not enough to deal with uncertainty. So, we explicitly represent uncertainty and appropriately taking it into account in forming a rule. The uncertainty is basically of two kinds: fact and rule. We deal with the factual uncertainty by creating new valuations for attributes. The rule uncertainty is semantically identical to the uncertainty of its conclusions. The uncertainty of attributes and conclusions are unified into the assertion uncertainty. Human experts tend to deal with uncertainty through qualitative symbols, or specifically linguistic words. Precisely, their description of assertion uncertainty is discrete. So, we can represent uncertainty by segmenting the quantitative description into qualitative one. For example, if the decision is about the weather, the basic classifications are clear, cloudy, rain. We have:

the weather is
1. clear 2. cloudy 3. rain.

When we are not sure it will rain, a number, called credit, will be associated with rain. We first quantify it into maybe, mostly, certainly. The new values are added in to replace "rain", i.e.,

3a. maybe rain
3b. mostly rain
3c. certainly rain.

In this way, the above basic representation is transformed into a 3–tuple:

(OBJECT ATTRIBUTE EXTENDED VALUE),

which is the right way of representing an assertion for examples in IR1.

At the end of rule formation, certainty of an assertion is again separated away. That is, the certainty about rule is explicitly represented in IR1 and this is consistent with human cognition. In fact, at the generalization step(see Section 4.3.3) of IR1, an effort is made to remove certainty(or called credit) from rules. A numerical rule will be generated to compute the credit of a conclusion from credits of conditions. So, a rule is a logical combination of assertion, which, is represented in predicate logic.

IR1 also has represented a concept framework hierarchically. This is to describe the generalization and specialization relation between concepts. Also any object must belong to a specific concept in order to form a predicate as in Step 1c of Section 4.3.3.

In the following section of the paper, examples are still written as an attribute–value pair to simplify writing.

4.3. Empirical Components

A group of new algorithms are created in IR1 to perform empirical tasks. They make IR1 distinguished from other empirical as well as integrated learning systems. Two features are shared by IR1's empirical components. First, they are designed to operate both incrementally and in batch, depending arguments supplied. In the case of incremental running, arguments supplied must be able to determine the scope of rules to be modified. Second, any comparison or computation must be done with credit. The examples belonging to the same class must have the same credit. Now, let us outline them briefly.

4.3.1. Generating Classification Rules with CLASS

The procedure CLASS is similar to PRISM. The difference lies in their processing the equal priority of two or more attribute–value pairs. PRISM constructs its rules by selecting an attribute–value pair one by one. The selecting sequence determines the simplicity of the rules. The selecting standard is to compute the priority for each attribute–value pair (Cendrowska, 1987). The equal priority for two or more pairs happens sometimes. IR1 adopts a thorough search which will guarantee the simplicity of the rules.

4.3.2. Establishing a Primary Hierarchy With EPH

We associate a number, called cost, with each selected attribute which is usually difficult or costly to be valuated and is called as undetermined attribute. The number, ranging from 1 to 100, indicates the availability of the attribute. We will use a simple example to illustrate the procedure ERH. Assume such a rule:

a2 & b3 & d2 & c4 = = > @1

If the cost of d and e is respectively 20 and 30, we will decompose the rule into:

a2 & b3 = = > TEST(d)
TEST(d) & a2 & b3 & d2 = = > TEST(c)
TEST(c) & a2 & b3 & d2 & c4 = = > @1,

where TEST means an action of implementing a test to valuate the corresponding attribute. In this way we establish a hierarchy: testing d, testing e and making decision. Assume d in Table 1 is an undetermined attribute, EPH may generate a few additional rules such as:

c2 & b1 = = > test(d)
a1 & c2 = = > test(d)
c1 & b2 = = > test(d)
c1 & a1 = = > test(d)
c1 & a2 = = > test(d).

4.3.3. Generalizing Rules With GR

Generalization was exhaustively investigated in AQ11 by Michalski and Stepp(1983). We borrow their approach with some modifications to fit our case. For instance, seeds are predetermined and it is unnecessary to select seeds. The resulting procedure, namely GR, is characterized as bellow:

Step 1. Stars are constructed.

In our case, a star G(@i) is defined as a set of all maximally general complexes (a logical description of a class) covering all the examples of class @i and not covering other examples in the training set. The construction begins with the rule obtained through above steps. The rules are generalized as much as possible by GEN operators until they cover the other examples. GEN operators partly come from AQ11 and partly are created by the authors. The resulting stars are not needed to be reduced as done in AQ11 in order to cover as much as possible from an incomplete training set.

1a. To linear selectors, the "closing the interval" rule is first applied. For instance, 1, 2, 3, 7, 8 is turned int0 1..3, 7..8 or 1..8 according to a threshold of how much generalization can be done. Specifically for integer selector, some "conceptual abstraction" rules are applied. For instance, 2, 4, 8, 16 is turned into square of an integer n. 4, 7, 10, 13 is turned into 3n+1.

1b. To structured selectors, the "climbing the generalization hierarchy" rule is applied. For instance, b1, b2, b34, b21 is turned into b, where b1, b2, b34 and b21 are the specific cases of the structure b.

1c. To any objects, the "predicate formation" rule is applied. A set of objects belonging to the same concept and sharing the same preconditions are merged into a variable appearing in a predicate. The predicate is usually corresponding to an attribute.

1d. To any selectors, the "Augmenting selectors" rule is applied: the enumerative forms of several selectors are combined into a relational form among them. For example, a = 1, 2, 5, 7, 12 and b = 3, 4, 7, 9, 14 is turned into a = b−2. The rule can be generally stated as searching for numerical regularity from data of the variables. This problem was exhaustively investigated by Wu(1988).

1e. After steps 1a, 1b and 1c, the "dropping the condition" rule is applied to all selectors: A selector is removed if this does not exceed the generalization threshold.

1f. To all classification rules, the "merging into a general common form" rule is applied. This is done on the basis of a belief that different classification rules should have some symmetrical forms. If two rules are not symmetrical, the specific one should be changed into the more general form in order to make them symmetrical. This substep is of some value for refining hierarchy.

1g. To all rules, credit computation rules are generated. Explicitly representing credits of attribute–pairs in rules will lead to too much specific rules. We try to eliminate credits from rules and generate special rules to compute credits of conclusions from credits of conditions. This just like the numerical discovery from a set of discrete data. Here, we use a simplified Reduction (Wu, 1988) to fin-

ish the task.

Step 2. Stars are optimized.
An optimized classification is built by selecting and modifying complexes from stars to make them mutually disjoint by procedure NID as in AQ11.

Step 3. A Primary Termination Criterion PTC is evaluated.
This criterion is based on AQ11 with some modification of the classification quality evaluation function LEF. The biggest difference lies in IR1's neglect of the sparseness--- a measurement of the difference between the input training set and the examples that the rules cover. This is because we want the induced rules as instructive as possible to deal with the more new cases.

4.3.4. Refining Hierarchy With ERH

Refining the primary rule hierarchy obtained in Step 3.3 is finished by abstracting similar selectors in all rules. A combination of some common selectors are grouped together to form an intermediate node in the hierarchy. This node is usually corresponded to a concept in the domain. Thus, we encourage domain experts to point out the important intermediate concepts to help form a refined hierarchy. Besides, IR1 also supports an automated formation of the refined hierarchy. Let us follow an example. Suppose the initial subset of the rules are as:

a1 & b1 & c1 & d1 = = > @1
a3 & b2 & c1 & d1 = = > @1
a1 & b1 & c3 = = > @2
a3 & b2 & c3 = = > @2
a4 & c2 & d3 = = > @2.

refining is done in a four-step procedure ERH(Establishing Refined Hierarchy):

Step 1. Merge rules into a disjoint form. The above subset are combined into:

(a1 & b1 v a3 & b2) & c1 & d1 = = > @1
(a1 & b1 v a3 & b2) & c3 = = > @2
a4 & c2 & d3 = = > @2.

Step 2. Replace the common sub-expression with a new concept introduced. In this example, the new concept I is defined as a boolean combination of a and b. That, is:

a1 & b1 v a3 & b2 = = > I1
a4 = = > I2
I1 & c1 & d1 = = > @1
I1 & c3 = = > @2
I2 & c2 & d3 = = > @2.

Step 3. Split the combined rules. The purpose is to simplify the rules into:

a1 & b1 = = > I1
a3 & b2 = = > I1
a4 = = > I2
I1 & c1 & d1 = = > @1
I1 & c3 = = > @2
I2 & c2 & d3 = = > @2.

Step 4. The Hierarchy Termination Criterion HTC is evaluated. If the hierarchy quality is not satisfied, go back to Step 2 to consider other common sub-expressions until the quality is no longer improved. The quality, being computed through HTC, is determined by four factors:

4a. Simplicity of a single rule. The long rule (involving many selectors) is usually considered as a poor one and needs decomposition by establishing intermediate concepts(nodes). On the other hand, too short rules are not welcomed either.

4b. The number of possible valuations of the introduced concept. The less valuations will make the rules for classifying the new concept are relatively simple and thus of great significance.

4c. The number of the levels of the hierarchy. The new concepts are encouraged to form other new concepts. The number must also be proportional to the number of the rules. This is because we want a tree whose branches are comparable to its depth. In a graphical term, the tree should be neither too "wide' nor too "deep".

4d. The number of the total rules. The less rules are extremely encouraged.

4.4. Analytical Issues

Let us go back to Fig. 2. Empirical components of IR1 first generate a set of inference rules which are also to act as the domain theory for explanation-based learning. When a new example comes in, the rules involving variables are invoked to explain the example. The explanation is also viewed as problem solving and explanation

result is the solution to the example.

According to the explanation result, different strategies are adopted. The analytical component, i.e., explanation–based generalization, only happens when having a perfect explanation(or a workable solution to the example problem). New domain rules are to be generated by generalizing the explanation structure. Mitchell et al(1986) presented a standard explanation–based generalization technique: modified goal regression. We first implemented their model without much modification and named it as MEBG(Mitchell's EBG).

However, as DeJong and Mooney(1986) pointed out, Mitchell's EBG is just one part of what explanation–based learning means. They specifically emphasize structural generalization. In fact DeJong(1988) has a more systematic view on what kinds of generalization can be done for EBL. Structural generalization involves changing the internal structure of the explanation. This action is more open(not well guided or constrained) and its result is of greater significance. So, we also implemented our second analytical component DMSG(DeJong and Mooney's Structural Generalization).

Our primary working domain is the hepatitis diagnosis. The hepatitis comprises Type A and Type B. We originally hoped that rules for two types could be learned from each other. That is, DMSG is, in some sense, an analogical learning. In fact, Ellman(1989) has argued that EBL can be viewed as analogical learning. This emphasis makes its learning less blind and relatively easy. In fact, the current version of IR1 implements Carbonell(1983)'s transformational analogy. Of course, we are also ready to have more complicated explanation–based generalizations, such as derivational analogy(Carbonell, 1986), number generalization and temporal generalization(DeJong, 1988).

5. Experiments

To have a flexible architecture, each component of IR1 was implemented as an independent module and can be run in batch(on all rules and classes) or incrementally(on selected rules and classes). Components can be arbitrarily combined to accomplish the designed task. Two common combination is an IR1 in batch and an incremental IR1. This flexible architecture also facilitates debugging greatly because each module stores its output into a file which reveals intermediate results to developers and users.

Our first experiment is the problem stated above. Given a training set as in Table 1, IR1 generated the desired results described in the last section. When given a subset of training examples, Incremental IR1 also generated the desired results as other examples input in. An interesting phenomenon is that the rules can be derived without inputing example No. 17, 18. IR1's GR module succeeded in updating its rules to include Rule No.7 as in Fig. 1. No variables were generated for rules and analytical learning was not verified in this experiment.

The second experiment intended to verify IR1 in a complex real–world context. We chose hepatitis diagnosis as a working domain, partly because its epidemic sometimes leads to a disaster and partly because it is typical of intermediate concepts or testing an unknown attribute. An appropriate rule hierarch sharing many commonalities with doctor's diagnosis rule is generated from a set of medical cases(exemplary patients with diagnosis) and refined from accumulative new cases. More interesting results appeared in its analytical learning components. After MEBG was applied, it was ironically found that new rules generated by MEBG are just ones putting the intermediate conclusions together. In other words, new rules have no hierarchy for the final diagnosis. That MEBG just goes back to PRISM's classification rule, with variables added. In our context, this is not welcome. Fortunately, we found that DMSG is quite useful. It successfully generated a few rules for Type A hepatitis diagnosis from explaining Type B hepatitis diagnosis rules.

A point should be made about this experiment. We originally aimed to use empirical components to set up the domain theory to escape EBL's strong dependence on a perfect domain theory. We had hoped only inputting a set of training instances, probally with a conceptual framework. However, the domain theory generated by IR1's empirical components are not enough to play a role of justifying the new rules from making an analogy with the old one. The justification needs a support from the medical theory. So, we also embedded some basic medical theoretical knowledge into IR1 to have an explanation–based analogy.

6. Concluding Remarks

The paper introduces an integrated frame work of inducing expert rules from a large set of examples and refining rules from accumulative examples. Its greatest feature is a mutual integration of empirical and analytical learning techniques. Its learning behavior is very similar to that a human being acquires knowledge.

Facts(examples) are collected and analyzed into a theory(rules) which is then updated or verified by new facts gradually(incrementally). Also its output (rule) is also similar to expert rule in a number of dimensions. Last, a new group of learning algorithms are presented to accomplish these goals. All of these efforts make it close to automated rule acquisition for KBSs.

However, when stepping to a practical rule learning system, we inevitably find some obstacles. The most serious one is that empirical as well as analytical learning, especially generalization, needs theoretical guidance. This theory is a theory about rule generation, or precisely meta-theory. In the incremental case, IR1's empirical components are directed by its explanation results. However, this is just an indication of where the refining should be made, not of how the refining is done. The same problem also happens to its useful analytical component DMSG, which also needs to be directed on how structural generalization is done. Currently, DMSG only conducts a simple structural mapping. More complex generalizations still stand away from a practical approach. For man, the theory corresponds to his general knowledge, or precisely common sense knowledge or world knowledge. To our dismay, incorporating common sense into machine has been and will continue puzzling AI researchers.

If, one day, we fortunately find some ways of common sense reasoning. We still cannot escape a dilemma: where and how the common sense knowledge comes from. We use an integration of SBL and EBL in order to escape EBL's dependence on existing domain theory, but we ironically fall into a trap where an existing common sense theory is required.

More practically, we feel a demand of making more experiments to verify this complicated framework. We are planning to use data which have been used by other learning systems to have a direct comparison. We also think of extending IR1's explanation-based generalization to generate more significant rules. In one word to summarize our approach, much has been done and more is left to future.

ACKNOWLEDGEMENTS

This research is supported by National Natural Science Foundation under Grant No. 68905004.

REFERENCES

Carbonell, J.G.(1983). Learning by Analogy: formulating and generalizing plans from past experience. in: Michalski, R.S., Carbonell, J.G. Mitchell, T.M.(eds), *Machine Learning: an artificial approach, vol 2*.Morgan Kaufmann.

Carbonell, J.G.(1986). Derivational Analogy: a theory of reconstructive problem solving andexperience. expertise acqisition. In: Michalski, R.S., Carbonell, J.G. Mitchell, T.M.(eds), *Machine Learning: an artificial approach, vol 2*. Morgan Kaufmann.

Cendrowska, J.(1987). PRISM: An Algorithm for Inducing Modular Rules. *Int. J. Man-Machine Studies*, Vol.27, pp.349-370.

Danyluk, A.P.(1987). The Use of Explanations for Similarity-Based Learning. *IJCAI-87*. Milan, Italy:Morgan Kaufmann.

DeJong, G.F.(1988). Some Thoughts on The Present and Future of Explanation-based Learning. *ECAI-88*.Munich, FRG: Pitman Publishing.

DeJong, G.F. Mooney, R.(1986). Explanation-based Learning: an alternative view. *Machine Learning*.Vol 1, No. 2, pp.145-176.

Ellman, T.(1989). Explanation-based Learning: a survey of programs and perspectives. *ACM Computing Survey*.Vol 21, No. 2, pp.163-221.

Kedar-Cabelli, S.(1987). Formulating Concepts According to Purpose. *AAAI-87*.Seattle, WA:Morgan Kaufmann.

Lebowitz, M.(1986). Integrated Learning: controlling explanation. *Cognitive Science*,Vol. 10, No. 2, pp. 219-240.

Mitchell,T,M.,Keller, R., & Kedar-Cabelli, S. (1986). Explanation-based Generalization: a unifying view. *Machine Learning*,Vol.1, No.1, pp. 47-80.

Pazzani, M.(1988). Integrated Learning with Incorrect anf Incomplete Theories. *ICML-88*. Ann Arbor, MI:Morgan Kaufmann.

Quinlan, J.R.(1983). Learning Efficient Classification Procedures and Their Application to Chess End Games. In Michalski, R.S. et al(Eds.) *Machine Learning: an artificial intelligence approach*. Morgan Kaufmann.

Quinlan, J.R.(1986). Induction of Decision Trees. *Machine Learning*, Vol.1, No.1, pp. 81-106.

Quinlan, J.R.(1987). Generating Production Rules From Decision Trees. *IJCAI-87*. Milan, Italy:Morgan Kaufmann.

Michalski, R.S. & Stepp, R.E.(1983). Learn-

ing From Observation: conceptual clustering. In Michalski, R.S. et al(Eds.). *Machine Learning: an artificial intelligence approach.* Morgan Kaufmann.

Schlimmer, J.C. and Fisher, D.(1986). A Case Study of Incremental Concept Induction. *AAAI–86.* Philadelphia, PA:Morgan Kaufmann.

Swaminathan, K.(1989). A Model of Integrated Learning. *Knowledge–Based Systems,*Vol. 2, No. 2.

Utgoff, P.E.(1988). ID5: an Incremental ID3. ICML– 88. Ann Arbor, MI:Morgan Kaufmann.

Wu, Y.H.(1988). Reduction: a Practical Mechanism of Searching for Regularity in Data. *ICML–88.* Ann Arbor, MI:Morgan Kaufmann.

LANGUAGE LEARNING

Adaptive Parsing: A General Method for Learning Idiosyncratic Grammars

Jill Fain Lehman
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Natural language interfaces have fallen short of their goal of providing full freedom of expression in human-computer interactions largely because significant increases in the coverage of the grammar are usually accompanied by intolerable decreases in system performance. We demonstrate a method for overcoming this barrier called *adaptive parsing*, a technique in which the system accommodates the user by dynamically growing its grammar to acquire her preferred forms of expression and recognize them directly in the future. The specific learning method in an implemented adaptive parser is discussed in detail and shown to be adequate to acquire eight idiosyncratic grammars with good coverage *and* good performance.

1. Introduction

The philosophy behind the design of natural language interfaces is to permit the user the full power and ease of her usual forms of expression to accomplish a task. A pragmatic consideration at odds with this philosophy is that we can neither write down a full grammar for English nor anticipate all ungrammatical or idiomatic utterances that may occur in spontaneously generated input. As a result, the fundamental decision in most interface designs is the choice of a restricted subset of English to constitute the recognizable grammar.

In choosing a sublanguage, an interface designer is faced with a dilemma. A small grammar with minimal ambiguity has the advantages of fast processing and single interpretations for most accepted utterances, but the disadvantages of poor coverage and brittleness (accepting, for example, "Schedule a meeting at 5 pm" but not "Add a 5 pm meeting to my schedule" or even "Schedule meeting 5 pm"). As the size of the sublanguage grows, coverage is increased but processing time and the number of interpretations produced for each utterance tend to grow as well. Further, these negative factors increase without any guarantee that the extensions made to the grammar conform to the needs of the particular user who must endure the performance degradations.

Solving this dilemma requires that we be able to choose the right sublanguage for each user. Since such a choice cannot be made *a priori*, we must rely on the interface to accommodate the user's idiolect automatically. In the next section we introduce a model of language learning able to acquire an idiosyncratic grammar through repeated experience with a user. In Sections 3 and 4 we discuss a particular implementation of the model called CHAMP. In Section 5 we demonstrate that the system's recognition capability and response time asymptotically approach near-optimal performance given any stable inherent ambiguity in the user's grammar.

2. The Model

The process of adaptive parsing is presented in Figure 1. Part (a) shows four user utterances (u1-u4) the first of which lies within the language recognized by the current kernel grammar (K) and the remainder of which do not. To say that u1 is grammatical, or *non-deviant* with respect to K, means that there is a set of grammatical forms in K that map the utterance to an appropriate meaning structure directly. Analogously, u2 through u4 are ungrammatical, or *deviant* with respect to K, because there are no such sets of forms. Thus, a static interface with restricted sublanguage L(K) will accept u1 but reject u2, u3, and u4.

Part (b) demonstrates the search for a meaning structure for deviant input. When an utterances lies outside of K, we extend failed parse paths by applying a *general recovery action* to a deviation with respect to a grammatical form. There are four types of recovery action corresponding to the four types of deviation for which they compensate: insertion, deletion, substitution, and transposition. Note that the set is complete—every utterance can be mapped into a set of grammatical forms by zero or more applications of these actions.

As an example corresponding to Figure 1(b), consider u2 = "Arrange a meeting 5 pm" and a grammar K in which postnominal cases must be marked by valid prepositions and "arrange" is unknown. A parse using K alone (Deviation-level 0, or simply, Level 0) will reject

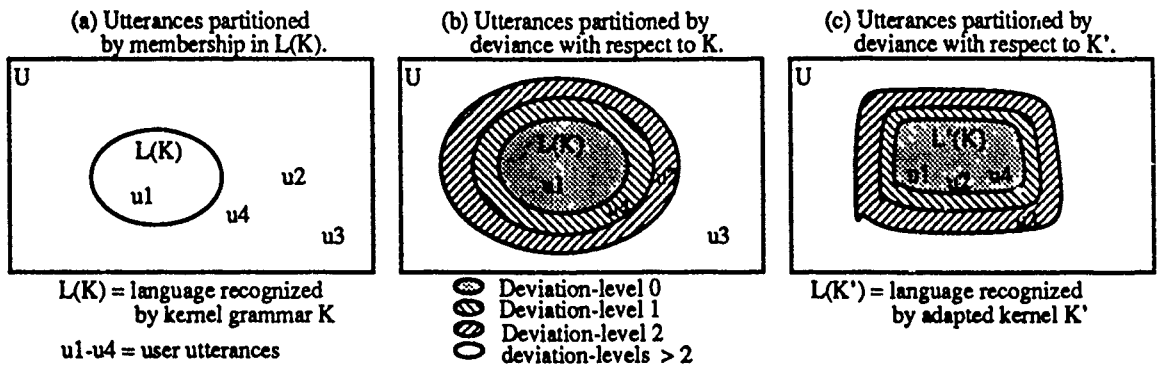


Figure 1: Adaptive Parsing. In response to u2, an utterance outside of L(K), the search for a meaning is extended beyond L(K) in a least-deviant-first manner. Understanding u2 results in adding new forms to K, creating the adapted kernel K'. Since the new forms recognize generalizations of the deviations in u2, other utterances in the user's language space may be less deviant with respect to K' than they were with respect to K.

u2. By extending the parse to allow one recovery action we may either interpret "arrange" as a substitute for a recognizable verb, or account for the deletion of the postnominal marker, but not both on the same path. Thus, the parse will fail at Level 1 as well. Allowing two recovery actions along a path, however, we find a mapping from the forms in K to a meaning structure for u2.

As the example indicates, we extend the search in a *least-deviant-first* manner, exploring grammatical interpretations first. Then, if no meaning is found, paths containing a single recovery action are considered, then two recovery actions, and so on. The use of a general, composable method of recovery applicable at any point in the input distinguishes error recovery in our model from previous relaxation techniques that restricted recovery to specific transformations on specific constituents in the grammar [2] [5] [9] [11].

Another way in which our model differs from previous techniques is that deviations do not remain deviations. Figure 1(c) represents the adaptation process. During adaptation the current kernel is augmented with new grammatical components that capture the general form of the deviations described by a meaning structure and its recovery actions. The ability to learn more than one new grammar component from an utterance, as well as the ability to learn both lexical definitions and syntactic forms, distinguishes our method of adaptation from previous methods that could learn only a single component and, in general, only a single type of linguistic knowledge [1] [3] [4] [8] [10] [12].

If we arbitrarily choose a sublanguage for a non-adaptive interface there is no guarantee that the trade-off between coverage and performance is to any particular individual's advantage—additional forms may contribute to ambiguity in the search space without corresponding to the user's preferred forms of expression. In an adaptive environment the trade-off still exists; adaptation may bring ambiguity into the current kernel. But if the individual tends to rely on previously accepted forms of expression such increases may still be advantageous. In prior work we demonstrated that although users differ significantly from each other in their preferred forms of expression, with frequent use they show regular, *self-bounded* linguistic behavior—a tendency to rely on those forms that have worked in the past [6] [7]. This behavioral regularity means that once accepted, an idiosyncratic form is likely to be reused. The more often it is reused, the more advantageous adaptation becomes; by modifying the grammar to recognize a deviant form directly, subsequent encounters with that deviation will not require error recovery. In addition, future sentences that share the learned deviation will be interpretable at lower deviation-levels, requiring less search.

Figure 1 shows adaptation's bootstrapping effect graphically. Assume an implementation of the model with a limit on search of two deviations (some limit is needed to enforce reasonable system response and preclude nonsensical interpretations). Also assume that each of u2, u3, and u4 uses "add" as its verb. Part (b) shows that u4 is interpretable at Level 1 but that u3 would be outside L(K) even with error recovery. Once u2 is brought into L(K), however, the deviances of u3 and u4 are defined in relation to the adapted kernel K': u4 is no longer deviant (having been brought into L(K') with u2) and u3 is now reachable through recovery. Thus, in an adaptive interface

all three utterances are accepted without rephrasing, and only u2 and u3 require search beyond Level 0. In a system with error recovery alone, u2, u3, and u4 would require extended search (to accept u2 and u4 and reject u3), and u3 would require rephrasing. In a static interface with restricted sublanguage L(K), each of u2, u3, and u4 would be rejected, requiring additional search for at least one rephrasing per utterance.

3. CHAMP, an Adaptive Interface

The model described above has been embedded in a working interface named CHAMP (CHAMeleonic Parser) which is implemented in COMMON LISP on an IBM RT. In addition to learning idiosyncratic grammars, CHAMP's task is to aid a user in maintaining a schedule of events and assist in arranging airline reservations for events that require travel. The number of general actions and object types in the system is fairly small (about fifty), although the number of specific objects, such as particular names and locations, is unconstrained. A full description of CHAMP can be found in [6].

The system uses a bottom-up, semantically and pragmatically constrained, least-deviant-first parsing algorithm. CHAMP performs deviation detection and recovery in terms of the four general recovery classes (insertion, deletion, substitution, and transposition), allowing at most two deviations in an utterance. Deviations are detected with respect to syntactic *forms* and lexemes in the current grammar. A form is a declarative structure specifying the requirements for assigning a segment of the utterance to a grammatical category.

To understand CHAMP's parsing algorithm, let us again consider the sentence "Arrange a meeting 5 pm." Simplified versions of two forms used in understanding the sentence can be seen below:

```
ACT1 isa addform
strategy:
  701 m-hourform
  702 m-intervalform
  703 m-dateform
  704 addword
  708 m-i-ggform
required: 704 708
unordered: 701 702 703
exclusive: 701 702
```

```
HR0 isa m-hourform
  111 hourmarker
  112 hourform
required: 111 112
```

The first form, ACT1, recognizes references to adding an entry to the calendar. ACT1 requires that the input contain at least two contiguous constituents: a verb in the class **addword** and an indefinitely marked group-gathering (**m-i-ggform**). The form allows the verb to be preceded by prepositional phrases describing a time slot (by start hour or interval but not both) and a date, in either order. HR0 is the kernel form that recognizes the marked hour (**m-hourform**) called for in step 701 of ACT1. HR0 requires as contiguous constituents a preposition (**hourmarker**) and an unmarked hour.

Figure 2 shows a portion of the least-deviant-first parse for "Arrange a meeting 5 pm." For clarity, we include only those constituents that are part of the final parse tree. The bottom of the figure shows the first step in parsing: segmenting the tokens in the input according to the class information attached to the words and phrases in the lexicon. Thus, "pm" is mapped to **nightword** while "arrange" is mapped to the special class **unknown**. A word or phrase may belong to more than one class; a unique leaf node is created for each meaning.

After segmentation, the bottom-up parsing algorithm is run at Level 0. In our example, most of the work done at this level consists of assigning leaf nodes to strategy steps that call for a member of the leaf's class. The only constituent constructed at this time is an unmarked hour; it is created by binding the **number** "5" to step 4 and the **nightword** "pm" to step 9 in HR1.

Since no complete parse could be constructed at Level 0, CHAMP continues the search by allowing a single deviation along any path in the tree. In our example, tolerating a single grammaticality causes a series of events. First, a marked hour is constructed via HR0 by binding the previously built unmarked hour to step 112 and allowing a deletion deviation of the required, missing step 111. A deletion annotation records the recovery. Next, the marked hour is attached to the **meetingword** to create a group-gathering via GG1. The group-gathering is then attached to the **indefmarker** to create an **m-i-ggform** via IGG1. This constituent may serve as the direct object of an **add** action (step 708 in ACT1). For the parse to succeed with only one deviation, however, ACT1's other required step (704) must be satisfied with a known member of the class **addword**.

Since no known **addword** occurs in the input, a complete parse tree cannot be constructed and CHAMP must continue at Level 2. Since two deviations are now permitted, the system may combine the indefinitely marked group-gathering with the unknown lexeme "arrange" to form a complete **add** action. A substitution annotation records the recovery.

The meaning structure produced by the parser is called an *annotated parse tree*, or APT. As seen in Figure 2, an APT contains each of the following types of information (the *adaptation context*): the particular grammatical forms

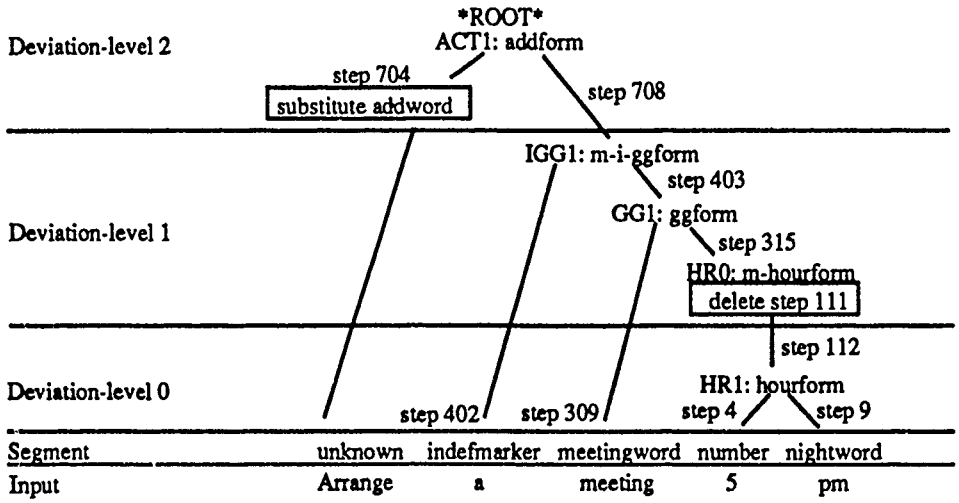


Figure 2: The annotated parse tree (APT) constructed for "Arrange a meeting 5 pm."

used to recognize the constituents, the order in which the constituents appeared in the sentence, and the recovery actions used to modify one or more of the forms. Every annotated parse tree must meet the pragmatic constraints represented by the contents of the calendar and airline databases (in our example, the five p.m. slot for the current day must be empty). In addition, an APT for a deviant utterance must have its projected effect on the calendar confirmed by the user (in our example, the user would be shown the new calendar entry and asked if it should be added to the schedule). Once the meaning has been established, deviation annotations must be converted into new grammatical constituents through adaptation.[1]

4. Adaptation and Generalization in CHAMP

The purpose of adaptation is to bring a deviant form into the grammar by deriving new grammatical components from the adaptation context that will parse the deviation directly in future utterances. The particular set of new components that are added to the grammar depends upon which deviations are present in the utterance. As shown in Figures 3 and 4, CHAMP learns new lexical definitions in response to substitution and insertion deviations, and new syntactic forms in cases of deletion and transposition.

The figures reveal that CHAMP learns discriminations based on the presence, absence, or position of categories in its kernel grammar. Since none of the adaptations introduces new constituents into a derived form, however, some portions of the user's natural language may be out of reach. Consider the following sentences taken from one user's first session with CHAMP:

s20: change june 11 ny to pgh from flight 82 to flight 265
s21: change from flight 82 to flight 265 on june 11 ny to pgh
s22: change flight 82 to flight 265 on june 11

This user's first two attempts to perform the task require learning that is beyond the scope of CHAMP's adaptation mechanism. The problematic segment is "from flight 82" which the parser tries to explain as a source in the **change** action's source-target pair because of the marker "from." No kernel form permits a flight object in the source (only flight attributes such as location), and the system provides no way to introduce the possibility into the language. Thus, both s20 and s21 are rejected. S22, which does not contain the misleading marker, is parsed by a kernel form. Note that the limitation here is in CHAMP—the model places no restrictions on inserted or substituted constituents.

In adapting to an identified deviation, we want to construct a set of new grammatical components with two properties. First, they must be accessible during future parses to understand this sentence and others like it directly. Second, the new components must not add unduly to the cost of understanding sentences in which they ultimately play no role. In short, the main issue in adaptive parsing, as in most kinds of learning, is the issue of generalization: capturing the correct conditions on usage of the deviation.

[1]CHAMP produces all parses at a deviation-level. If different APTs correspond to different effects, then identifying the intended effect may also disambiguate the parse. If a number of APTs all correspond to the correct effect (because of true structural ambiguity in the grammar), then all are passed onto adaptation where new grammatical constituents are created as *competitors*. A competition is resolved by a future utterance that requires some constituents but not others.

Substitution

- **Adaptation**: a new lexical definition is added to the grammar for the unknown word or phrase as a member of the word class that required the substitution. *Example*: "Arrange a meeting at 5 pm" where "arrange"is unknown results in the addition of "arrange" to the lexical class **addword**.
- **Generalization**: occurs through class references in the grammar. Everywhere the word class was previously sought, the new lexeme is recognized.
- **Learning**: permits substitutions of words and phrases only. Extends the set of indices for discriminations already present in the grammar through the word class. Loses potential discriminations based upon the tokens themselves, or the co-occurrence of the tokens or class with other classes or deviations present in the adaptation context.
- **Search**: reduces the deviation-level required to understand future occurrences of the lexeme. If the lexeme was already defined in the lexicon, the new definition increases the amount of lexical ambiguity in the system. This may create additional search paths for utterances that contain the lexeme, but has no affect on the search space for utterances that do not contain it.

Insertion

- **Adaptation**: a new lexical definition is added to the grammar for the unknown word or phrase as a member of the special class **insertword**. *Example*: "Please cancel flight #451" adds "please" to **insertword**.
- **Generalization**: occurs throughout the grammar. The new lexeme is allowed to occur without deviation anywhere in subsequent utterances.
- **Learning**: assumes the lexeme carries no meaning. Permits insertion of words and phrases only. Loses potential discriminations based upon the co-occurrence of the tokens with other classes or deviations present in the adaptation context.
- **Search**: same as for substitution.

Figure 3: Summary of adaptation to substitutions and insertions.

With respect to accessibility, Figures 3 and 4 show that a derived component that recognizes a new way of referring to a class of constituents is "inherited upward" through the grammar. In other words, the new component can be used by all forms that call for constituents of that class. The main advantage to using class inheritance for generalization is that it provides a simple, uniform mechanism. The disadvantage is undergeneralization; learning across established boundaries in the grammar requires discrete episodes. Consider, as an example, that the kernel distinguishes group-gatherings marked by indefinite articles from those marked by definite articles by assigning the former to the class **m-i-ggforms** (which can be part of an **add** action) and the latter to **m-d-ggforms** (which cannot). The first time a user drops an article from her utterance, she does so in one context or the other; CHAMP learns a new instance of one of the classes, but not of both.

With respect to cost, two factors are relevant: overgeneralization and ambiguity. Although inheritance upward can overgeneralize the correct conditions on usage, bottom-up parsing algorithms tend to compensate for this fairly well. Ambiguity is more problematic. As Figure 3 reveals, substitution and insertion adaptations may introduce lexical ambiguity if the new definitions act as alternatives to entries already in the lexicon.[2] The cost of the ambiguity during search depends initially upon the number of additional forms indexed by the new definition. The degree to which the ambiguity propagates through the search space depends upon what other constraining information is provided by the utterance.

Deletion adaptations may introduce structural ambiguity into the grammar. Let us reconsider the case of **m-i-ggforms** and **m-d-ggforms**. In the kernel there is only one member of each class, DGG1 and IGG1. Both forms require a marker and a subconstituent from the class **ggforms**. The *critical difference* between the forms is the marker class; as long as each form requires a different marker, both forms cannot succeed at the same deviation-level. If the user drops both definite and indefinite articles, however, the system will derive forms omitting each marker; both DGG1' and IGG1' will require only one subconstituent, an unmarked group-gathering. Viewed differently, any unmarked group-gathering object will succeed at Level 0 in the adapted grammar as a member of both classes. Again, the cost of the ambiguity during search depends initially upon the number of additional forms indexed by the incorrect class assignment. The

[2] In CHAMP, it is impossible to introduce a new lexical definition for a word or phrase that is already defined without employing extra-linguistic conventions. The *lexical extension problem* is discussed in [6].

degree to which the cost propagates depends upon the other constituents in the sentence.

Transpositions may introduce ambiguity if the critical difference between the parent and the derived form is in an ordering relation on constituents not present in the utterance. Consider the example in Figure 4. ACT1 and ACT1' differ by the ordering relation imposed on **addword** and **m-dateform**. Thus, if no date appears in an utterance recognizable by ACT1, the utterance will also be recognizable by ACT1'. Because of an optimization in the implementation, this sort of adaptation only introduces ambiguity at non-zero deviation-levels, although the ambiguity always propagates through the search space.

It is important to note that even though each type of adaptation may increase the amount of ambiguity in the current grammar, the future search space for utterances containing the deviation is always significantly smaller than it would have been had we not adapted. The reason is simple: any ambiguous paths at Level 0 were also part of the much larger search space at the higher deviation-level. Of course, the cost of parsing some utterances that were in the grammar prior to the adaptation will have increased, but three factors make the trade-off worthwhile. First, since adaptations reflect preferred forms of expression, utterances relying on the adaptations are likely to reappear; the more often an adaptation is reused the more favorable the trade-off becomes. Second, adaptation brings more of the user's language into the grammar, resulting in fewer rejected parses over time. Since rejecting a sentence requires a search through Level 2 *plus* the search associated with understanding any subsequent rephrasings, any action that prevents rejections must reduce search overall. Finally, since the frequent user's language is self-bounded, whatever increase in ambiguity results from adaptation must be bounded as well.

5. Analysis of the Utility of Adaptation

The purpose of this section is to provide a sense of the overall utility of adaptation and generalization by answering two questions for CHAMP's performance on users's spontaneously generated input. First, what is the

Deletion

- **Adaptation:** a new form is added to the grammar. It is derived from the form in which the deviation occurred and inherits all the information present in its parent that does not relate to the deleted step. *Example*: "Arrange a meeting 5 pm" results in the creation of HR0' derived from HR0. HR0' is also a member of the class **m-hourforms** but its strategy and required lists do not contain **hourmarker**.
- **Generalization:** occurs through class references in the grammar. The derived form may be used anywhere the parent form is used to recognize a constituent. Deletions and transpositions within a single constituent are preserved as co-occuring.
- **Learning:** permits future discriminations based on the presence or absence of a class. Permits discriminations based upon some co-occurrences of deviations.
- **Search:** reduces the deviation-level required to understand future utterances containing the deletion. May introduce structural ambiguity into the grammar if the deleted step represents the sole difference between two grammatical categories.

Transposition

- **Adaptation:** a new form is added to the grammar that explicitly captures the new ordering of steps. The new form inherits all the information present in the parent that does not relate to the ordering of the transposed step. The ordering relations on the transposed step depend on the steps surrounding it after transposition. *Example*: "Schedule on June 4 a meeting with Alice" results in the creation of ACT1' from ACT1. ACT1' has the strategy (m-hourform m-intervalform addword m-dateform m-i-ggform) where only the hour and time interval are unordered.
- **Generalization:** occurs through class references in the grammar. The derived form may be used anywhere the parent form is used to recognize a constituent. Transpositions and deletions within a single constituent are preserved as co-occuring.
- **Learning:** permits discriminations based on ordering. Permits discriminations based upon some co-occurrences of deviations.
- **Search:** reduces the deviation-level required to understand future utterances that reflect the new ordering. If the transposed step is not required, the adaptation may introduce ambiguity into the search space at non-zero deviation-levels whenever the transposed step is not needed to understand the utterance.

Figure 4: Summary of adaptation to deletions and transpositions.

	User 1	User 2	User 3	User 4	User 9	User 10
G_{kernel}	18/127 (14%)	34/144 (24%)	9/138 (7%)	25/130 (19%)	20/212 (9%)	19/173 (11%)
G_{final}	115/127 (91%)	127/144 (88%)	112/138 (81%)	118/130 (91%)	192/212 (91%)	148/173 (86%)

Figure 5: The effectiveness of learning as measured by the differences in the number of utterances understood by the kernel and by the user's final grammar.

effectiveness of learning? That is, how much actual improvement in understanding the user's language comes from adaptation? Second, what is the cost of learning? That is, what effect do adaptation and generalization have on the level of ambiguity in the grammar?

The data used during evaluation came from two sources. In early experiments, Users 1, 2, 3, 4, 5, and 7 interacted with a simulated adaptive interface based on the model described in Section 2 (Users 6 and 8 were in a non-adaptive control condition). Of the 657 utterances collected from these users, 85 were chosen from Users 1 and 2 to guide in the design of CHAMP. Users 9 and 10 participated in subsequent, on-line experiments with CHAMP, producing an additional 385 test utterances. In both experiments the user's task was to look at a pictorial representation of a change to the calendar and use the system to effect that change in the on-line schedule. Since Users 5 and 7 did not complete all nine experimental sessions, we consider here only results for the six users who did.[3]

What is the effectiveness of learning in CHAMP? To answer this question we contrast for each user the number of her sentences accepted by the original kernel and the number accepted by her final grammar—any increase is due to learning. Figure 5 shows that the increase is significant for each user and that CHAMP's performance on data collected during the simulation experiments did not differ significantly from the system's performance on data from the on-line experiments. On the average, the kernel accepts only 16% of a user's utterances while her own derived grammar accepts 88%.

To carry the analysis further, we measure the utility of each learning episode by computing the average number of sentences brought into the grammar each time a deviant utterance is interpreted. These values (3.3, 2.7, 2.3, 3.7, 5.7, and 5.6) represent a kind of "bootstrapping constant" that reflects the way in which CHAMP's particular implementation of adaptation captures within-user consistency. An alternative implementation would probably produce very different values. Consider, for example, an interface using a more conservative approach to substitutions by creating a new class for the substituted word and a new form requiring that class (unlike CHAMP, such a system permits discriminations based on the tokens themselves). The tendency of this approach to undergeneralize is likely to appear as a decrease in the utility of each learning episode: the user's final grammar might result in fewer acceptances, or in the same number of acceptances but at the cost of requiring more instances of adaptation. Thus, the values themselves are not as important as the fact that our ability to compute them provides a metric for comparing design choices.

The second question we posed was: what is the cost of adaptation? As CHAMP brings more of the user's language into the grammar it increases the likelihood that it will understand her future utterances. But is the increase in understanding, as measured by acceptances, negated by a larger increase in the cost of understanding, as measured by search? We know that the user's language is self-bounded, but it may still be quite ambiguous. The question, then, is not whether we can prevent the rise in search stemming from inherent ambiguity in the user's idiolect, but whether the system as a whole suffers disproportionately as ambiguity increases.

We measure the rise in ambiguity in an adapted grammar in two ways. First, holding the test sentences constant, we compare the average number of parse states considered by successive grammars during search at Level 0. As the amount of ambiguity in the grammar increases through the adaptations of each session, so will the

[3]Because of differences between the model and the implementation some relatively minor preprocessing of utterances from the simulation experiment was done prior to evaluation. In addition, extra-grammatical markers were introduced to compensate for the problem of lexical extension in both groups (see previous footnote). For a full description of the experiments, a list of the test utterances (with modifications indicated), and a number of other evaluation results, see [6].

User	Number of utterances accepted w/o deviation		Average number of states to accept		Average number of parse trees/utterance	
	G(0)	G(9)	G(0)	G(9)	G(0)	G(9)
U1	18	115	19.3	37.6	1.1	1.5
U2	34	123	21.9	59.8	1.1	1.6
U3	9	112	18.0	43.3	1.0	2.0
U4	25	118	21.0	40.7	1.1	1.5
U9	20	192	13.0	80.9	1.1	2.6
U10	19	148	40.4	45.8	1.1	1.7

Figure 6: Change in the cost of parsing non-deviant utterances as a function of grammar growth.

average amount of search required to accept an utterance at Level 0. Second, we examine the average number of APTs produced for each sentence accepted at Level 0. This value reflects a rise in ambiguity that is partly independent of the increase in search because of the way APTs share substructure. A rise in search need not give rise to additional parse trees. Similarly, additional APTs may indicate only a modest increase in search. From the user's point of view increased search corresponds to decreased response time while a rise in the number of parses corresponds to a rise in the number of interactions required for resolution of her intended meaning.

Figure 6 summarizes the relevant measures for each user by showing the values for the kernel grammar (G(0)) and for her adapted grammar at the end of session nine (G(9)). For four of the six users, the increase in the number of parse states examined is proportionally far less than the increase in the number of sentences understood. By the end of the ninth session, the search for User 1 has expanded by a factor of two but the number of her sentences that are now non-deviant has expanded by a factor of six. User 3's trade-off is even more favorable: twelve times as many sentences are accepted by the final grammar as by the kernel, at a cost of only two and a half times the search. User 4 gains almost five times as many sentences at slightly less than twice the search. The trade-off is most favorable for User 10 who gains almost eight times as many accepted sentences with virtually no increase in search. User 2 has the most balanced case: a factor of 3.5 increase in accepted utterances and a factor of three increase in search. The largest increase in search occurs for User 9 (about a factor of six) but the ten-fold increase in the number of her sentences accepted still places her within the general trade-off ratios seen among the others.

If increase in search corresponds to increase in response time, what do these values tell us? In short, response times will get slower over all but will not grow exponentially as a function of the increase in the language accepted. Since the grammar itself is bounded by the user's natural behavior, the increase in response time is bounded as well. Even with the kinds of increases in search seen for these users, response times were usually under ten seconds.

The near-monotonic increase in the size of the search goes hand-in-hand with a near-monotonic increase in the average number of APTs produced for each accepted utterance. Where is the ambiguity coming from? The users do introduce some lexical ambiguity into their idiosyncratic grammars, but in CHAMP's parser lexical ambiguity contributes primarily to small, local increases in the size of the search space. Examination of the parse trees produced with successive grammars for the same sentence showed that most of the increase in ambiguity comes from adaptation to deletion deviations. As the user's language becomes increasingly terse, the system builds forms in which the content words that correspond to critical differences between strategies are deleted. As a result, there is an increase in the number of constituents that are satisfied by each segment; often the increase propagates to the root nodes themselves. User 9 is a case in point: her grammar included derived forms omitting almost every content word in the kernel. Specifically, by the end of session three she had dropped most markers, three of the four verbs, and two of the four group-gathering head nouns. As a result, almost every sentence she typed in the last seven sessions created at least two APTs. On the last day the simple sentence, "Dinner June 24 with Allen," created twelve parse trees at Level 0.

6. Summary and Future Work

In total, CHAMP has been tested on 1042 utterances most of which represent unmodified spontaneous input by frequent users whose job includes calendar scheduling as part of its duties. We found no qualitative differences between CHAMP's performance for utterances gathered by simulation of the model and the system's performance for utterances produced during on-line interactions. We

did find that acceptance of user utterances relied significantly on adaptation. On average, the kernel grammar accepted 7% to 24% of each users utterances, while her final grammar accepted 81% to 91%. We also found two areas of system performance that could be improved.

First, although CHAMP was able to understand about 84% of each user's utterances, this rate is about 8% lower than that predicted by the model for the test sets. The difference is caused primarily by deviations that require the system to introduce a new type of meaningful constituent into a context that does not expect it. An unrestricted implementation of insertion and substitution recoveries would solve the problem, although such an extension would have non-trivial implications for learning. A single insertion, for example, can usually be attached at many locations in the parse tree, any of which may represent the correct or most useful generalization of the utterance's structure. It is unclear whether the existing competition mechanism is adequate to eventually determine the most useful form. If not, what heuristics might we use to decide?

Second, although CHAMP's adaptations resulted in only a modest rise in ambiguity in each adapted grammar, the increase is only partly explained by inherent ambiguity in the user's idiolect. A significant portion of the added ambiguity comes from the interaction between critical differences and adaptation to deletion deviations. Kernel forms were designed to contain critical differences with respect to each other because such differences help to constrain search. Yet, whenever adaptation eliminates a difference, ambiguity may result. Thus, a possible method for overcoming this source of performance degradation is to track critical differences explicitly and create a new grammatical category whenever a discrimination that separates two existing categories disappears. Such an extension raises interesting questions: when do you replace existing categories, and when do you simply augment the grammar with a new category? In the case of augmentation, which previously existing forms should refer to the new category and which to the old?

Regardless of the improvements offered above, evaluation of the system clearly shows that adaptation is a robust learning method. CHAMP was able to learn eight very different grammars, corresponding to very different linguistic styles, with a single general mechanism. We believe this represents a significant step forward in the effort to provide users with full freedom of expression in computer interactions.

Acknowledgments: The clarity and content of this paper were improved by the comments of Angela Kennedy Hickman, Richard Lewis, Thad Polk, and two anonymous reviewers.

References

1. Berwick, R.C., and Pilato, S., "Learning Syntax by Automata Induction," *Machine Learning*, Vol. 2, 1987.
2. Fain, J., Carbonell, J. G., Hayes, P. J., and Minton, S. N., "MULTIPAR: A Robust Entity-Oriented Parser," *Proceedings of the Seventh Annual Conference of The Cognitive Science Society*, 1985.
3. Granger, R. H., "FOUL-UP: A Program That Figures Out Meanings of Words from Context," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 1977.
4. Kaplan, S. J., *Cooperative Responses from a Portable Natural Language Data Base Query System*, PhD dissertation, University of Pennsylvania, 1979.
5. Kwasny, S. C., and Sondheimer, N. K., "Ungrammaticality and Extra-grammaticality in Natural Language Understanding Systems," *17th Annual Meeting of the Association for Computational Linguistics*, 1979.
6. Lehman, J. Fain, *Adaptive Parsing: Self-extending Natural Language Interfaces*, PhD dissertation, Carnegie Mellon University, 1989.
7. Lehman, J. F., and Carbonell, J. G., "Learning the User's Language, A Step Towards Automated Creation of User Models," in *User Modelling in Dialog Systems*, Wahlster, W., and Kobsa, A., eds., Springer-Verlag, 1989.
8. Miller, P. L., "An Adaptive Natural Language System that Listens, Asks and Learns," *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, 1975.
9. Minton, S. N., Hayes, P. J., Fain, J. E., "Controlling Search in Flexible Parsing," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1985.
10. Salveter, S. C., "On the Existence of Primitive Meaning Units," *18th Annual Meeting of the Association for Computational Linguistics*, 1980.
11. Weischedel, R. M., and Black, J. E., "Responding Intelligently to Unparsable Inputs." *American Journal of Computational Linguisitics*, Vol. 6, No. 2, 1980.

A Comparison of Learning Techniques in Second Language Learning

Steven L. Lytinen and Carol E. Moon
Artificial Intelligence Laboratory
The University of Michigan
Ann Arbor, MI 48109

Abstract

We present a machine learning program, called ANT, which learns the grammar of a second language. Input to the system is similar to what is found in a typical introductory foreign language text; that is, a mixture of instructions describing grammar rules, and examples illustrating these rules. We compare ANT's learning to two alternatives: learning from only instructions, and learning from only examples. We discuss why, from a *functional* or processing standpoint, learning from a mixed input is more effective than either of the alternatives. We also present an empirical comparison of our algorithm's performance on input containing both instructions and examples vs. performance of the system when given instructions only or examples only. The results of the comparison support our hypotheses as to the utility of mixed input.

1 Introduction

This paper describes a program called ANT (Acquisition using Native-language Transfer), which learns the grammar of a second language. ANT successfully learns approximately 85% of the grammar rules presented in a typical first-year German textbook. Input to the system is similar to what is found in a typical introductory text. The system modifies its English grammar rules accordingly, so that they correspond to the grammar of German. ANT can then "understand" German sentences.

Most all foreign language texts follow the same format when presenting a new grammatical construction. When one looks into a typical text, one does not find a list of instructions which explain the grammatical constructions of the second language. Nor does one find simply a list of example sentences which illustrate the foreign language's grammar. Texts almost always contain instructions and examples integrated together. The general format is the introduction of a new rule with an instruction, followed by a set of examples which illustrate the rule taught in the instruction. Why does this seem to be the optimal format for such a text, or at least the most common format?

In this paper, we will address this question from the standpoint of a machine learning theory. We present our theory of learning from instructions (which include examples), which has been implemented in ANT. We then discuss why, from a *functional* or processing standpoint, learning from this sort of input is more effective than either obvious alternative, learning from a set of examples, or learning from instructions without any examples. In building ANT, we have discovered several reasons why, computationally, it is helpful for our program to be given instructions along with examples in order to learn new grammar rules.

To provide further evidence in support of our theory, we present an empirical comparison of three types of learning: ANT's approach, which uses both examples and instructions; a version of ANT which learns from instructions only; and a version which learns from examples only. In terms of both efficiency and correctness, the original version's performance is superior to the performance of the two alternatives.

2 An example of ANT's performance

To explain how ANT learns, we begin by presenting an example lesson. While there are differences in ANT's processing depending on what type of rule it is learning, this example illustrates the main points of how ANT works. The example lesson is the following.

> In German, verbs come at the end of relative clauses.
>
> Examples:
>
> > Der Erdferkel, der Ameisen oft frißt,
> > läuft langsam
> > (the aardvark who ants often eats
> > runs slowly)[1]

[1] ANT does not receive an English translation as part of its input. The literal English translation is provided here for the benefit of the reader.

Der Mann, der mir Bücher gibt,
wohnt in Paris.
(the man who me books gives
lives in Paris)

ANT's grammar rules are written in a unification-style format (Shieber, 1986), as discussed in (Lytinen and Moon, 1988; Moon and Lytinen, 1989). However, for the purposes of this paper, we can assume that they are context-free rules.

Before receiving this input, ANT assumes that its English relative clause (RC) rules apply to German.[2] Two of these rules are the following:

(1) RC → RP VG NP
(2) RC → RP VG NP NP

Some of ANT's rules for VG's (verb group) and CONJ-V (conjugated verb) are given below:

(3) VG → CONJ-V ADV
(4) VG → CONJ-V
(5) VG → CONJ-V INF

(6) CONJ-V → AUX
(7) CONJ-V → MODAL-AUX
(8) CONJ-V → V

ANT produces the following representation when it reads the instruction:

```
ORDER
    CONSTITUENT :  CONJ-V
    OCCURS-IN : RC
    POSITION  :  LAST
```

This means for ANT that the rule to be learned has to do with ordering, specifically with the position of the conjugated verb in the relative clause.

Modifying ANT's English relative clause rules to conform to German is not a straightforward matter. This is because the context-free rules for relative clauses (RC) do not even mention verbs. The constituent VG appears in them, but simply moving the VG to the end of each RC rule would not be the correct modification, since the verb is not always the last constituent in a VG. Thus, ANT cannot simply change its RC rules. Other relevant rules, such as the rules for what makes up a VG, potentially must also be changed.

This is where examples come into play in the learning process. Without examples, ANT would have to search through its grammar for possible appearances of verbs within relative clauses. In the worst case, this could mean searching the entire grammar, since a verb could in theory appear inside of any constituent of a RC, and RC's could possibly contain every other kind of constituent. In addition, we could not in general guarantee that the search would find the correct verb, as there could be (and in fact there are) several occurrences of VG and CONJ-V within relative clauses.[3] However, because ANT is provided with examples in addition to instructions, ANT lets the examples guide it to the rules which must be changed. ANT does this by parsing the examples. During the parse, ANT is forced to use the rules which must be modified for German. Thus, the potentially large search through the grammar is avoided.

Getting back to our example, because the rule ANT is learning is an ordering rule, ANT relaxes the ordering constraints in its relative clause rules when parsing the examples, meaning that it will be able to parse a sentence whose relative clause word ordering does not conform to English grammar.[4] Let us consider the first example:

Der Erdferkel, der Ameisen oft frißt, läuft
langsam
(the aardvark who ants often eats runs
slowly)

ANT parses this example, arriving at the parse tree in Figure 1. Once the example is parsed, it is clear that the verb "frißt" (eats) is the verb in the relative clause. Now ANT has identified the constituent that must move: "frißt" is the CONJ-V within the VG within the RC. This information tells ANT that the category VG must be modified, at least when VG's appear in relative clauses. This leads ANT to modify rule 1 from before to construct the following new grammar rules:

(9) RC → RP NP VCOMP CONJ-V
(10) RC → RP NP CONJ-V

Notice that the category VG has been eliminated from these rules. This is because the rules for VG must stay the same, since verb groups occur in other rules besides RC rules. In RC rules, VG is replaced by the conjucated verb (CONJ-V) along with a new category, called VCOMP, the vestiges of the VG category. The rules for VCOMP are as follows:

(11) VCOMP → ADV
(12) VCOMP → INF

Two RC rules (9 and 10) are generated because VCOMP is optional.

Processing of the second example helps ANT modify relative clause rule 2 in a similar way, creating the following final set of rules.

(13) RC → RP VCOMP NP CONJ-V
(14) RC → RP NP CONJ-V

[2] This is analogous to a well-known phenomenon in human second language learning, known as *native language transfer*, in which learners typically selectively transfer patterns from the native language to the second language (Selinker, 1969).

[3] For example, relative clauses contain NP's, which can contain other relative clauses, which contain verbs.

[4] The way in which ANT's unification grammar rules are written is essential to ANT's ability to parse sentences which do not conform to its grammar, and is beyond the scope of this paper.

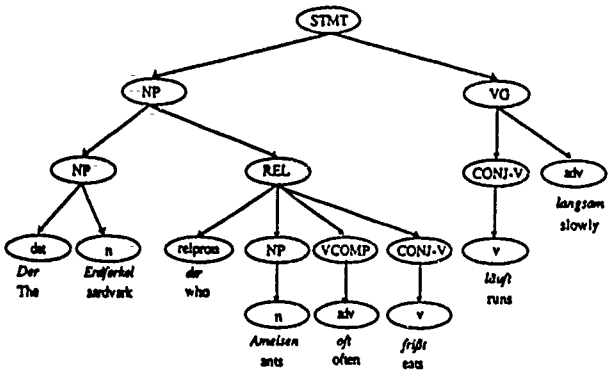


Figure 1: Parse of the aardvark example

(15) RC → RP VCOMP NP NP CONJ-V
(16) RC → RP NP NP CONJ-V

3 Why ANT needs instructions and examples

We have seen an example of how ANT processes input to learn a new grammar rule. The input consists of a mixture of instructions and examples. Now let us summarize the roles that the two parts of the input play in the learning process.

3.1 The role of instructions

When ANT learns a new rule, there are three distinct roles played by the instruction portion of the input:

- alter expectations of input
- focus attention on part of the example that illustrates the new information
- determine how general the new rule is

The actual formation of new rules takes place during the processing of examples. However, the way in which examples are processed is greatly influenced by the instruction. First, ANT's ability to parse German examples is facilitated by the instruction, because it tells the system something about the differences to be expected in the examples. In our relative clause rule, the instruction told the system that its word order constraints might be violated within example relative clauses. This enabled ANT to parse these examples even though they did not conform to its (English) grammar.

The instruction also improves the formation of new rules. From the information in the instruction, the system can focus attention on the important part of the example, and thus can more quickly find the change(s) to be made to the original rules. Often the instruction clearly indicates how general the new rules should be. In our relative clause example, the instruction told the system that any modifications to the category VG should only affect relative clauses. Without this information, ANT would have had no way of knowing whether all verb groups are different in German than they are in English, or perhaps some other subclass of VG's, such as only verb groups followed by a single NP. Many such hypotheses would be consistent with the examples that ANT received.

3.2 The role of examples

While instructions are critical to ANT's performance, the examples presented along with the instructions also play a crucial role in the system's learning process. In particular, examples play the following three roles:

- identify relevant previous knowledge
- form new rules
- fill in details not mentioned in instructions

The first role above was illustrated in the example in section 2. Rather than search its grammar to identify relative clause rules which had to be changed, ANT relied on examples to point the way to previous rules. The examples identified the location of the verb in the relative clause and thus gave the system the knowledge it needed to be able to relate RC to CONJ-V.

New rules to be learned are often formed as part of the parse of the example. After the parse is complete, part of the parse tree is an instantiation of the new grammar rule. Thus, ANT can extract the new rule directly from the parse of the example. However, the instantiation is often more specific than the new rule ought to be. The information from the instruction is used to determine how general the new rule can be made.

The third role is not illustrated by our relative clause example. However, often it is the case that instructions found in textbooks do not completely specify the rule to be learned. In these cases, examples must fill in the details left out of the instruction. Because of the way that ANT extracts rules from example parses, this is a natural by-product of ANT's learning process.

4 Learning from only instructions or only examples

It seems intuitive that learning should be easier if the learner is provided with both instructions and examples than with either alone. ANT's learning process suggests possible reasons for this. In learning from instructions without examples, ANT would have difficulties with the following tasks:

- identifying relevant previous knowledge
- relating terms to constituents

- inferring details not mentioned in the instruction

As we saw in the relative clause example, without examples to guide the search for relevant English grammar rules, finding these rules could potentially be very costly. In the worst case, finding relevant rules could mean inspecting the entire grammar. Determining the correct relevant rules may not even be possible if there are several alternatives from which to choose

Another difficulty is that ANT's internal representation of grammatical categories may not exactly match the terminology used in the instructions. For example:

> In German statements, the verb must be the second constituent.

It is not clear from this instruction what constitutes a "constituent." Without extensive further explanation, there are several possible interpretations of this instruction. For example, it is not clear from this instruction what the German equivalent of "In the park the man slept" would be. There are several possibilities (using English words instead of German):

> In slept the park the man.
> In slept the man the park.
> In the park slept the man.
> The man slept in the park.

The third choice above is the correct one, but determining this requires knowing what constitutes a constituent. Does a preposition alone qualify, or an entire prepositional phrase? Also, what is the ordering of the constituents after the verb? Without further explanation as to what the term "constituent" refers to, ANT cannot deduce what the correct German rule is.

Finally, even if terminology is not a problem, often instructions simply do not contain all the necessary information to infer the correct German grammar rule. For example:

> In German, the verb "haben" used with the adverb "gern" means "to like."

Many German textbooks leave out the details of this rule, such as where the object of "haben" should appear in the sentence (after the verb but before "gern"). This is more restrictive than general rules for placement of adverbs in German.

There is an obvious solution to the problem that instructions leave out details: one might simply insist that instructions be written so as to include details that are often left out. However, in designing a learning system, it is desirable for that system to be instructed in a way that is natural for people. Judging from language textbooks, instructors find it most natural to leave out some information from instructions, relying on examples to enable the reader to infer the details of the rule.

For ANT, learning from examples without instructions would also be problematic. In particular, the following tasks would be more difficult:

- parsing examples
- identifying relevant features
- generalizing new rules

As we stated before, the instruction tells ANT in what ways the examples will deviate from English grammar. Without this information, it is often possible to arrive at the wrong parse of an example. Without the correct structural analysis, ANT cannot induce the correct German grammar rules.

Even if the correct structural analysis is arrived at for an example, there may be other problems. The number of possible features on which to base an hypothesis is very large. Without instructions, ANT cannot know whether the point of the example is to show case agreement, a word order modification, a novel German construction for which there is no equivalent English construction, and so on. Thus, we would expect the learning process to be much slower if instructions were not included in the input.

Finally, even if the correct features are identified, there is the issue of how general the new rule is. Examples alone cannot always convey the correct conditions for the application of a rule. For example, in German there is a rule that the verb must be the second constituent in a statement. Suppose the system processes some examples of statements. It is plausible that the system (or a person) could realize the generalization to be made is that the verb is second, but not know whether this applies to all verbs (like those in clauses), to only the main verb in the sentence, to verbs in questions, and so on.

5 An empirical comparison

In order to demonstrate empirically that learning from instructions and examples is easier than learning from instructions only or examples only, we implemented versions of ANT which learn from instructions only and from examples only. We then compared performance of these versions on a small subset of the rules that original ANT learns.

5.1 Instructions-only ANT

The comparison between original ANT and the version which learned from instructions only focused on the amount of searching through the grammar that was necessary in instructions-only ANT. This search was not necessary at all in original ANT, since processing examples yielded the relevant English rules as a by-product of parsing.

Given knowledge of how to find an embedded constituent by searching the grammar, instructions-only ANT was given five reordering instructions to learn. These instructions were the following:

> (1) In a statement, the verb is in second position.

Instruction	Rules searched	Items searched
1	5	7
2	23	15
3	2	3
4	5	8
5	10	12

Figure 2: Performance of Instructions-only ANT

(2) When a dependent clause begins a sentence, the verb of the independent clause immediately follows the dependent clause.
(3) In a relative clause the verb comes at the end.
(4) In a dependent clause the verb comes at the end.
(5) When a modal auxiliary occurs in a statement, the infinitie comes at the end.

The instructions were randomly chosen from the set of reordering instructions that original ANT learns. Instructions-only ANT's task was to find the relevant English rule which would be affected by the instruction, and then change that rule.

ANT stopped its search as soon as it found any instance of the constituent it was looking for. For example, in the relative clause rule, ANT stopped as soon as it found any conjugated verb (CONJ-V) within a relative clause. This sometimes led to the incorrect selection of English rules to be modified.

Figure 2 shows the results for the five instructions. The number of rules examined is given, as well as the number of constituents, arrived at by summing the number of distinct consituents on the right hand side of each context-free rule that was searched. The average number of rules search is 9, about 20% of the size of the entire grammar.

Recall that the search terminated as soon as any instance of the desired constituent was found. Unfortunately, the wrong instance was found first for 3 of the 5 instructions. This suggests that the search should continue even after finding the desired constituent, to see if other instances of the constituent can be found. The result would be a much more extensive search. In addition, it is not clear how instructions-only ANT would be able to decide which instance of the constituent should be modified.

5.2 Examples-only ANT

In designing the examples-only version of ANT, many assumptions had to be made along the way, as there were no examples-only second language acquisition systems which we could directly compare to ANT [5]. We tried to be as generous as possible to the examples-only approach with these assumptions, greatly simplifying the task at times so as to be sure not to bias the comparison in favor of the instructions-and-examples approach.

One assumption we made was that transfer from the native language would be useful in the examples-only approach. Intuitively, it seems that learning should proceed faster if the system started with its knowledge of English rather than from scratch. Also, there is much evidence that people at least selectively transfer native language information to the learning of a second language. (Selinker, 1969; Kellerman, 1987; Anderson, 1983; Gass, 1980)

Another simplification we made was that only rules having to do with deviations from English word order would be learned. This meant the system had many fewer possible features to consider in the input examples when hypothesizing new rules; for example, it did not need to be concerned with agreement, case, etc. Next, part of the input to the examples-only version of the system was the parse of each of the examples. This greatly simplified the task of hypothesizing new rules, since the system did not need to guess at which constituents should be grouped together. The learning task, then, was to infer how general the rule should be which was illustrated by the set of examples and their parses. Finally, an assumption was made about the conditions for a new rule: it was assumed that the presence of a constituent, rather than its position or some other feature, would be the only conditions under which a new rule was required. For example, a rule such as "if the direct object is a personal pronoun, then it precedes the indirect object" could not be learned under this assumption, since its condition does not depend simply on the presence of a direct object, but on additional features of this constituent.

To learn from examples only, ANT performed the following analyses on the input it was given:

1. find all differences between the form of the rules used in the parse and the original English rules
2. find all descriptions possible of the ordering that occurs in the parse tree
3. find all possible conditions, derivable from the current example, under which the rule to be learned might be restricted, then find the intersection of all the conditions for all examples analyzed thus far
4. enumerate the descriptions which are in the intersection of all differences noted thus far for the set of examples and which are in the intersection of all descriptions found in (2).

These four steps are described below.

[5] One possible candidate for comparison was RINA (Zernik and Dyer, 1987), but this system learned mainly idiomatic expressions rather than basic grammatical constructions, and received more feedback from the teacher than the traditional examples-only paradigm of many machine learning programs.

Finding differences

To process the examples, ANT analyzes the parse trees for the examples (which it is given as part of its input), one at a time. The system looks at each node in the tree, in search of a grammar rule whose right-hand-side enumerates its children in the correct order. If such a rule cannot be found, then ANT looks for a rule which has the same constituents but not in the correct order. For example, let us assume the English rules for STMT are the following:

(17) STMT → PP NP VG NP
(18) STMT → NP VG NP

Now say that an example is presented, such as:

Im Park spielen die Kinder Fußball.
(In the park play the children soccer)

Since the construction of this example is PP VG NP NP, ANT first looks for a rule which matches this construction, STMT → PP VG NP NP. Since no such rule can be found, ANT picks STMT→ PP NP VG NP as the closest English rule. It then enumerates the differences between the rule and the ordering in the parse tree.

Finding the list of features

Next ANT needs to find all of the features, in terms of ordering, that exist in the tree, so that the common features across all examples are remembered. ANT not only should extract the features in its examples that show differences with the original rules. It also should extract the features common in all examples, saving those as candidates for the generalization it is trying to make. For example, when ANT works with the instruction about the verb coming second in the sentence, it may first receive an example that uses the rule 20 below:

(19) STMT → PP NP VG NP
(20) STMT → NP VG NP

In this case, ANT will not be able to detect any constituent ordering changes, since the original rule has the same constituent ordering as the example. However, all features of the example are potential candidates for generalization, depending on differences between English rules and subsequent examples in the example set. Say the next example in the set utilized rule 19 above. This time the ordering would be different, since the VG in the example would be second. One of ANT's hypotheses for generalization then would be that the VG is second in STMT. In order to verify that this feature is significant, and hence still a candidate for the generalization it should make, ANT must know if this particular feature occurred in all previous examples.

Finding conditions

The system must be capable of learning the conditions under which the changes that it learns apply. The range of possible conditions has been simplified by our assumption that the only conditions that can apply are those that involve the presence of a constituent in a particular category. Thus the system would be able to determine the condition in a rule to be learned like "When a modal auxiliary occurs in a sentence, the infinitive comes at the end," but it would not be able to detect the condition in "When a dependent clause begins a sentence, the verb of the independent clause follows the dependent clause."

Finding features common to all examples

After it is done processing an example, ANT combines the information from that example with the cumulative data gathered from examples thus far. It finds the intersection of differences found in the grammar and computes the union of that set with the intersection of what is found in all examples. In other words, ANT arrives at hypotheses for what the generalization of the rule might be both from what is in common with all previous examples and from what alterations of the original grammar were evident in the examples. The goal is to derive a minimal set of hypotheses, usually one, for what the general lesson is, where the general lesson is basically equivalent to the instruction in the instructions and examples version. Along with this generalization, the system should detect any relevant conditions.

5.3 Results

The examples-only algorithm was run on five sets of examples, and its performance was compared to the performance of the examples-and-instructions version of ANT on these rules. In the examples-only implementation, the correct rule was learned for four of the five sets, while one set of examples resulted in an incorrect generalization. The reason for this incorrect generalization is discussed below. Moreover, ANT was much slower at inducing the new grammar rule when using only examples. On average, the examples-only version required an average of 4.0 examples per rule learned, while an average of only 1.6 examples were required in the instructions-and-examples approach. Figure 3 compares the number of examples required to learn each rule for the two algorithms.

The rules used in the comparison were:

(1) The verb is the second constituent in a sentence.
(2) The verb comes at the end of the relative clause.
(3) The verb comes at the end of a dependent clause.
(4) When a dependent clause begins a sentence, the verb of the sentence follows the dependent clause.
(5) When a modal auxiliary occurs in a sentence, the infinitive comes at the end.

Rule number five was not correctly learned from examples only, in fact, given the assumptions that we

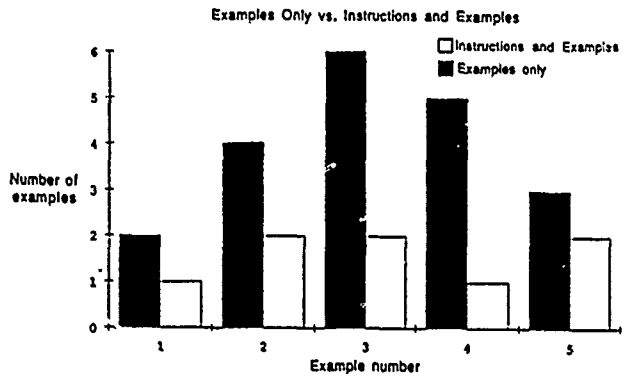


Figure 3: Comparison of Performance of Original and Examples-only ANT

made, no set of positive examples could eventually cause the system to derive the correct condition. This is because ANT cannot rule out from positive examples the alternative hypothesis that if an infinitive occurs in a sentence, then a modal auxiliary is also required. This alternate hypothesis is consistent with the data, and cannot be ruled out by positive examples which conform to rule 5.

6 Conclusion

We have presented a theory of learning which utilizes both instructions and examples to learn grammar rules of a second language. This theory suggests several functional roles for both instructions and examples in learning. Performance of our system declines substantially if it is required to learn from only instructions or only examples, both in terms of efficiency and correctness of learning. In learning from only instructions, efficiency declines in terms of the search required to determine what previous knowledge is relevant to the new information. In learning from only examples, efficiency is measured in terms of the number of examples required to learn a rule.

The discrepency in learning efficiency would potentially have been much greater if we had not made generous assumptions about our instructions-only and examples-only approaches. In both approaches, we assumed that all the rules learned would involve word order changes, as this vastly simplified the learning task. In the instructions-only approach, we allowed our system to terminate its search once any instance of the desired constituent was found. This often led to incorrect selection of English rules. In the examples-only approach, we assumed that all the rules would depend only on the presence or absence of a constituent in the sentence. In reality, many rules specify other aspects of grammar, or depend on other features, position of a constituent, case, or other features. Relaxing these assumptions would mean that the space of possible hypotheses to consider in deriving a rule would be vastly larger in either appraoch. This, in turn, would increase the amount of grammar search or the number of examples required to derive a new rule. In the examples-and-instructions version of ANT, however, these restrictions were not imposed. Thus, the efficiency comparison presented here is most likely biased against the examples-and-instructions approach.

Acknowledgements

This research was supported in part by a grant from the Kellogg Foundation under the Presidential Initiatives Fund at The University of Michigan.

References

Anderson, R. (1983). Transfer to somewhere. In Selinker, L., and Gass, S. (eds), *Language Transfer and Language Learning.* Newbury House Publishing, Rowley, MA.

Berwick, R. (1985). *The Acquisition of Syntactic Knowledge.* MIT Press, Cambridge, MA.

Gass, S. (1980). An investigation of syntactic transfer in adult second language learners. In Scarcella, R. and Krashen, S. (eds.), *Research in Second Language Acquisition.* Newbury House Publishing, Rowley, Mass., pp. 132-141.

Kellerman, E. (1987). Aspects of transferability in second language acquisition. Ph.D. Thesis, Katholieke Universiteit, Nijmegen, The Netherlands.

Lytinen, S., and Moon, C. (1988). Learning a second language. In *Proceedings of the Seventh National Conference on Artificial Intelligence,* St. Paul, MN, pp. 222-227.

Moon, C. and Lytinen, S. (1989). The function of examples in learning a second language from an instructional text. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society,* Ann Arbor, MI, August 1989. Lawrence Erlbaum Associates, Hillsdale, NJ.

Selinker, L. (1969). Language transfer. *General Linguistics 9,* pp. 67-92.

Shieber, S. (1986). *An Introduction to Unification-based Approaches to Grammar.* CSLI, Stanford CA.

Zernik, U., and Dyer, M. (1987). The self-extending phrasal lexicon. *Computational Linguistics 13,* pp. 308-327.

Learning String Patterns and Tree Patterns from Examples

Ker-I Ko[1]
Dept of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794

Assaf Marron[2]
Dept. of Computer Science
University of Houston
Houston, Texas 77004

Wen-Guey Tzeng[1]
Dept. of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794

Abstract

A (string) pattern is a non-null string over an alphabet and a set of variables. A pattern language is the set of all strings obtained by substituting non-null constant strings for variables in a pattern. We investigate learning pattern languages in three new directions. First we prove that to decide whether there is a string pattern consistent with given positive and negative samples is *NP*-hard. Thus it is not polynomial-time learnable under Valiant's probabilistic learning model unless $RP = NP$.

Then we discuss extensions of examples and string patterns: incomplete examples and tree patterns. We prove that to decide whether there is a common k-variable string pattern for given *incomplete positive* examples is *NP*-complete for any fixed integer $k \geq 2$. We give polynomial-time algorithms to find common k-variable tree patterns for non-associative, non-commutative constant trees for any fixed integer $k \geq 1$. We also prove that to decide whether there is a common k-variable tree pattern for associative, commutative constant trees is *NP*-complete for any fixed integer $k \geq 2$.

1 Introduction

A *(string) pattern* p is a non-null string over a constant alphabet Σ and a set X of variables. The *pattern language* $L(p)$ defined by a pattern p is the set of all strings over Σ which can be obtained by substituting non-null constant strings for variables in p, for example, $L(10xx01) = \{10\,00\,01,\ 10\,11\,01,\ 10\,0000\,01,\ 10\,0101\,01,\ 10\,1010\,01,\ 10\,1111\,01, \ldots\}$. In this paper we are concerned with learning patterns and certain generalized patterns from examples. Our model of learning is based on the exact-information, worst-case complexity model, in contrast to Valiant's probabilistic-information, probabilistic-complexity, distribution-free model [Va84]. Some results on learning pattern languages in Valiant's model can be derived as consequences of our results through the work of Blumer et al. [BEHW89].

Learning of patterns was introduced by Angluin [An80]. She gave a polynomial-time algorithm (in fact, a nondeterministic log-space algorithm) for finding the longest one-variable pattern from sample strings (or, from positive examples only). Following the principle of Occam's razor [BEHW87], the longest pattern will guarantee that the solution is fittest in the sense that no other suitable pattern language is a proper subset of it. For $n \geq 2$, the problem of finding the longest two-variable pattern from positive examples is left open. Ko and Hua [KH87] showed that a straightforward generalization of Angluin's algorithm for the two-variable case does not work in polynomial time unless $P = NP$. It suggests that this problem may be *NP*-complete. For the case where the number of variables is not fixed, even the simple membership problem for pattern languages (i.e. given a pattern p and a constant string s, determine whether $s \in L(p)$) is known to be *NP*-complete [An80].

We attack this problem in three new directions. First, we consider the problem of learning patterns from both positive and *negative* examples. More precisely, the new problem FCP of *finding common patterns from both positive and negative examples* is: given two sets S and T of constant strings, decide whether there exists a pattern p consistent with S and T, i.e., $S \subseteq L(p)$ and $T \subseteq \overline{L(p)}$.

We explain this problem by a possible application. Molecular biologists gather some sequences of nucleotides of some segment of DNA which is corresponding to some kind of disease. Some sequences are from normal people and some are from patients. Then they try to find (learn) a genetic pattern which can explain this phenomenon. There are two interpretations of gathered data. The first one is that a sequence of nucleotides is normal if it can be obtained from a genetic pattern by substituting some nucleotides for some specific positions, otherwise it is

[1]The work of these two authors was supported in part by NSF grant CCR-8801575.

[2]Current affiliation: IBM.

abnormal. In this case positive examples are those normal sequences and negative examples are those abnormal sequences. For example, if the genetic pattern AGxCxAT is learned from normal sequences AGTCTAT and AGGCGAT, and abnormal sequences AGGCAAT and AGTCAAT, then AGACCAT is an abnormal sequence of nucleotides which may cause a disease, where $\{A, C, G, T\}$ are four types of nucleotides. The second one is that an abnormal sequence of nucleotides has some specific sub-sequences of nucleotides. In this case, the positive examples are those abnormal sequences and negative examples are those normal sequences. A sequence is abnormal if it can obtained from the learned genetic pattern. This is because the specific disease sub-sequences will appear as sub-sequences of the learned genetic pattern. For example, the genetic pattern AGxATyC is learned from abnormal sequences AGCATTC and AGAATGC, and normal sequences ACTTCAC and AGCAGGC, then AGTATGC is an abnormal sequence. Our problem corresponds to this learning procedure, where nucleotides correspond to Σ. For the first interpretation of data, the specific positions of the genetic pattern correspond to the variables of our pattern. For the second interpretation, the specific sub-sequences correspond to constant sub-strings of our patterns. With this tool, scientists can tell whether an unborn baby has a risk of some kind of disease through a genetic examination.

We show that the problem FCP is *NP*-hard when the number of variables in p is not fixed. An interesting consequence of this result is, from the general result of Blumer et al., that learning patterns in Valiant's probabilistic learning model cannot be done in polynomial time unless $RP = NP$. Kearns and Pitt [KP89] also considered learning pattern languages in this direction. They gave a polynomial-time algorithm for learning k-variable pattern languages, for any fixed $k \geq 1$, under arbitrary *product* distribution. However, their algorithm outputs a set of simpler patterns with k or less variables instead of one single k-variable pattern. Thus, the precise complexity of learning k-variable patterns from positive and negative examples and outputting a single k-variable pattern is still left open for any fixed $k \geq 1$. This problem falls into the complexity class *NP* since the membership problem of k-variable patterns is in P for any fixed $k \geq 1$. Recently, Schapire [Sc89] also showed that pattern languages are not learnable in Valiant's learning model if $P/poly \neq NP/poly$. However, he allows empty-string substitution of variables. This model is different from ours that prohibits empty-string substitution. It appears that this slight difference does affect the complexity of the learning problem.

The second approach is to learn patterns from *incomplete* examples. This problem can also apply to the above genetic pattern problems. It is not unusual that experiment data are not perfect. They may miss some parts of genetic sequences and thus we may have incomplete samples.

We say a string $s' \in \Sigma^+$ is *consistent* with a string $s \in (\Sigma \cup \{?\})^+$ (assuming $? \notin \Sigma$) if $|s'| = |s|$ and for every i, $s'(i) = s(i)$ whenever $s(i) \neq ?$, where $s(i)$ denotes the ith character in s. Let p be a pattern. A string $s \in (\Sigma \cup \{?\})^+$ is an *incomplete positive* example for p if there exists an $s' \in \Sigma^+$ such that s' is consistent with s and $s' \in L(p)$. The question here is to determine whether incomplete examples make learning more difficult. Our result supports an affirmative answer: for any fixed $k \geq 2$, the problem of finding the longest k-variable pattern from incomplete positive examples is *NP*-complete (while, as discussed above, the corresponding problem using complete positive examples is still not known to be *NP*-complete). However, the 1-variable case of the problem remains open (while, the 1-variable case of complete positive examples is polynomial-time solvable [An80]).

The third approach is to generalize the string patterns to two-dimensional *tree patterns* and adding the *commutativity* property to the concept of patterns. The idea of tree patterns comes from the area of term rewriting of theorem proving [BKN85] [VR89]. A function f which corresponds to an internal node of a tree is *commutative* if $f(a, b) = f(b, a)$, where a and b are two subtrees of node f. A function g is *associative* if $g(a, g(b, c)) = g(g(a, b), c)$. A tree pattern is a tree with function symbols associated with internal nodes and constants or variables on leaves. A tree pattern generates constant trees (trees with no variables) by substituting constant trees for each variable. The membership problem for tree patterns is to determine, for a given tree pattern t and a constant tree s, whether s can be derived from t by a substitution. This problem is called *term matching* in the area of term rewriting. Efficient algorithms have been developed for several variations of this problem (e.g. [BKN85]).

A string pattern can be viewed as a depth-1 tree pattern having the associativity property. Thus, removing the associativity property from a tree pattern would potentially simplify the learning problem, and adding the commutativity property would make it more difficult. Our results support this intuition. We show that learning a tree pattern without the associativity and commutativity properties can be done in polynomial time no matter whether the number of variables is fixed or not. On the other hand, we show that if a tree pattern is allowed to have the associativity and commutativity properties, then the learning problem is *NP*-complete for the two-variable case.

The above *NP*-completeness results (incomplete examples and tree patterns) seem difficult to be strengthed to the one-variable case. Intuitively, the one-variable patterns do not have complex structures for reduction proofs of *NP*-hardness. Its complexity is, intuitively, closer to the original two variable string pattern learning problem of complete positive examples.

2 Preliminaries

Let Σ be a finite alphabet and X disjoint from Σ be a countable set. Then Σ^* denotes the set of all finite (constant) strings over Σ and $\Sigma^+ = \Sigma^* - \{\textit{empty string}\}$. Elements in Σ are called constants. Elements in X are called variables. Let $s_1 s_2$ denote the concatenation of two strings s_1 and s_2, and s^m the concatenation of m s's. Let $\|S\|$ denote the number of elements in set S.

A *(string) pattern* is a non-null string over $\Sigma \cup X$. Let $|p|$ denote the length of pattern p, for example, $|x0x10yy| = 7$. A pattern p is called *k-variable* if there are no more than k distinct variables occurring in p. A pattern p is called *k-occurrence* if each variable in p occurs at most k times. Assume that p is an n-variable pattern with variables $x_1, \ldots, x_n$. Then $p[s_1/x_1, \ldots, s_n/x_n]$ denotes the string obtained from p by substituting s_i for each occurrence of x_i in p, $i = 1, 2, \ldots, n$. The language $L(p)$ is defined to be

$$L(p) = \{p[s_1/x_1, s_2/x_2, \ldots, s_n/x_n] : s_i \in \Sigma^+, 1 \le i \le n\}.$$

Note that if $s \in L(p)$, then $|s| \ge |p|$. Since we can check whether two patterns are the same up to variable naming, in the rest of this paper we neglect the naming difference of patterns.

The following two propositions are from [An80]. The first one states some properties of pattern languages. The second one, as we said before, shows that the membership problem of pattern languages is *NP*-complete.

Proposition 2.1 *The class of pattern languages is incomparable with the class of regular languages and with the class of context-free languages. The class of pattern languages is not closed under any of these operations: union, complementation, intersection, Kleene plus, homomorphism, or inverse homomorphism. It is closed under concatenation and reversal.*

Proposition 2.2 *The membership problem for pattern languages is* NP*-complete, i.e., the following problem is* NP*-complete: given a pattern p and a string $s \in \Sigma^+$, decide whether $s \in L(p)$.*

3 Learning Patterns from Positive and Negative Examples

3.1 Positive Examples Only

Angluin [An80] gave a polynomial-time algorithm to find the longest one-variable pattern for a set of strings (positive examples), while the complexity of the two-variable case is left open (cf. [KH87]). For the problem of deciding whether there exists a pattern (not necessarily longest) for positive examples only, we should have a parameter of pattern length, otherwise pattern x would be a suitable solution for most cases. Without restriction on the number of variables of target patterns, we can easily find, in deterministic log space, a common pattern for a set of strings with pattern length greater than an input parameter r, because we only need to check that the shortest string of the set is of length $r' \ge r$ and use $x_1 x_2 \cdots x_{r'}$ as the solution. However, if we also use the number of variables as a parameter, the problem becomes *NP*-complete.

Theorem 3.1 *The following problem is NP-complete: given a finite set $S \subseteq \Sigma^+$ and two integers n and t, determine whether there is an n-variable pattern p of length $\ge t$ such that $S \subseteq L(p)$.*

This is proved by a simple reduction from the membership problem without bounds on variables. We omit it here.

3.2 Positive and Negative Examples

In this section, we consider the problem of finding patterns (not necessarily longest) from positive and negative examples. The main result of this section is that without bounds on the pattern length and the number of variables, we can prove that the question of determining whether there is a pattern consistent with given positive and *negative* examples is *NP*-hard.

Theorem 3.2 *The following problem is* NP*-hard: given two finite sets S and T of strings, determine whether there is a pattern p which is consistent with S and T, in the sense that $S \subseteq L(p)$ and $T \subseteq \overline{L(p)}$.*

Proof. We reduce the 3SAT problem [GJ79] to this problem. Let $U = \{u_1, u_2, \ldots, u_n\}$ be the set of variables and $C = \{c_1, c_2, \ldots, c_m\}$ be an arbitrary instance of 3SAT. We need to construct two sets S and T such that C is satisfiable if and only if there is a pattern p consistent with S and T. We first consider the restrictive case that a consistent pattern must have length $\ge 3n$.

Construct the following positive examples s_i's and s_i''s, and negative examples t_j's, $0 \le i \le n$ and $1 \le j \le n$:

$$s_0 = r_1 r_2 \ldots r_n,$$

where $r_k = 111$, $1 \le k \le n$;

$$s_0' = r_1 r_2 \ldots r_n,$$

where $r_k = 000$, $1 \le k \le n$;

$$s_i = r_1 r_2 \ldots r_n,$$

where $r_i = 000$, $r_k = 111$, $1 \le k \le n$ and $k \ne i$;

$$s_i' = r_1 r_2 \ldots r_n,$$

where $r_i = 1100$, $r_k = 111$, $1 < k < n$ and $k \ne i$;

$$t_j = r_1 r_2 \ldots r_n,$$

where $r_j = 101$, $r_k = 111$, $1 \le k \le n$ and $k \ne j$.

Let $S = \cup_{i=0}^{n} \{s_i, s_i'\}$ and $T'' = \cup_{i=1}^{n} \{t_i\}$. Assume $p = p_1 p_2 \ldots p_n$, $|p_i| = 3$, $1 \le i \le n$, is a pattern of length $3n$ which is consistent with S and T''. From positive examples s_0 and s_i, we know that p_i contains 3 variables (no constants) and variables in p_i do not occur in the other p_j's if $i \ne j$. From the negative example t_i, the middle variable of p_i is equal to the first or the last one of p_i. By s_i', p_i needs to match

1100, 100, or 110 to obtain s'_i, so p_i cannot be $x_i x_i x_i$. Thus $p_i = x_i x_i y_i$ or $x_i y_i y_i$.

For each clause $c_i = \{l_{i_1}, l_{i_2}, l_{i_3}\}$ in C, $1 \le i \le m$, we add t'_i as a negative example to guarantee that at least one literal in c_i is assigned to *true*, i.e., not all of them can be assigned to *false*. For $1 \le i \le m$, we define

$$t'_i = r_1 r_2 \dots r_n,$$

where

$$r_k = \begin{cases} 100 & \text{if } l_{i_j} = u_k, j = 1,2,3 \\ 110 & \text{if } l_{i_j} = \overline{u}_k, j = 1,2,3 \\ 111 & \text{otherwise.} \end{cases}$$

Let

$$T' = T'' \cup (\cup_{i=1}^{m} \{t'_i\}).$$

Now we want to show that a truth assignment τ satisfies C if and only if there is a p consistent with our examples S and T'. For the forward direction, we define $p_i = x_i x_i y_i$ if $\tau(u_i) = true$ and $p_i = x_i y_i y_i$ if $\tau(u_i) = false$ for $1 \le i \le n$. We can easily verify that $p = p_1 p_2 \cdots p_n$ is consistent with S and T'. In particular, for each cause c_i, if u_k occurs in c_i and $\tau(u_k) = true$, then $p_k = x_k x_k y_k$, but $t'_i = r_1 r_2 \cdots r_n$ with $r_k = 100$, and so $t'_i \notin L(p)$; similarly, if $\overline{u}_k$ occurs in c_i and $\tau(u_k) = false$, then $p_k = x_k y_k y_k$ and r_k in t'_i is 110, and $t'_i \notin L(p)$.

For the backward direction, if there exists a pattern $p = p_1 p_2 \cdots p_n$ of length $3n$ consistent with S and T', we define $\tau(u_i) = true$ if $p_i = x_i x_i y_i$ and *false* if $p_i = x_i y_i y_i$. Then for each clause $c_i = \{l_{i_1}, l_{i_2}, l_{i_3}\}$, $1 \le i \le m$, the negative example t'_i guarantees that one of p_{i_j}, $1 \le j \le 3$, makes $\tau(l_{i_j}) = tr$.

Last, we need to prove that no patterns of length $< 3n$ can be consistent with S and T. We add the following strings to T:

$$\{1^i : 1 \le i \le 3n-1\},$$

i.e., let

$$T = T' \cup \{1^i : 1 \le i \le 3n-1\}.$$

If some shorter pattern q is consistent with S and T, then all its positions must be variables since s_0 and s'_0 are positive examples. Since the length of q is less than $3n$, $1^{|q|}$ can be obtained as a positive example by substituting string 1 for every variable. This contradicts our negative examples. □

Remark. Note that by Proposition 2.2, the above problem is not known to be in *NP*. We showed that the complexity of this problem is complete for Σ_2^p with respect to log-space many-one reduction, where Σ_2^p is the class of languages recognized by nondeterministic oracle Turing machines in polynomial time relative to oracle sets in *NP*, i.e., the second level of the polynomial time hierarchy. It will be reported in a subsequent paper [TK90].

The above proof does not seem applicable to the case when the number of variables is fixed. Thus, the problem of finding a k-variable pattern, $k \ge 1$, which is consistent with positive and negative examples remains open.

Valiant [Va84] discussed a probabilistic learning model. In his model, examples, when requested, are provided according to some fixed, but unknown probability distribution. Learning algorithms are required to output an approximate concept using polynomial number of examples in polynomial time. Blumer et al. [BEHW89] showed a general result that the polynomial learnability of a concept under Valiant's learning model can be reduced to the consistency problem of the concept. That is to say, the approximate learning of Valiant is equivalent to the worst-case learning if the VC-dimension of the domain grows polynomially with respect to some size measure of the domain. In pattern languages, if we take the length of a pattern as the natural size measure, then its VC-dimension grows linearly by a simple counting argument. As a consequence of this result, we have

Corollary 3.3 *Pattern languages are not polynomial-time learnable under Valiant's learning model unless* RP = NP, *where* RP *is the class of problems which can be solved by randomized polynomial-time algorithms.*

4 Learning Patterns from Incomplete Examples

In this section we consider generalized examples, i.e., incomplete examples, for patterns. With this generalization, we can prove that learning the longest two-variable pattern from incomplete positive examples is *NP*-complete. However, the 1-variable case of the problem remains open.

Assume $? \notin \Sigma$. We say a string $s' \in \Sigma^+$ is *consistent* with a string $s \in (\Sigma \cup \{?\})^+$ if $|s'| = |s|$ and for every i, $s'(i) = s(i)$ if $s(i) \neq ?$, where $s(i)$ denotes the ith character in s. A string $s \in (\Sigma \cup \{?\})^+$ is an *incomplete positive* example for a pattern p if there exists an $s' \in \Sigma^+$ such that s' is consistent with s and $s' \in L(p)$. We first observe that the complexity of the membership problem for incomplete examples and k-variable patterns remains the same as that of complete examples for any fixed $k \ge 1$.

Theorem 4.1 *For any fixed $k \ge 1$, there is a polynomial-time algorithm for the following problem: given an incomplete example s and a k-variable pattern p, decide whether s is an incomplete positive example of p*

Proof: (Sketch) Let $|s| = n$. Assume the number of occurrences of x_i in p is n_i, $1 \le i \le k$ and the number of constants in p is n_0. Let s_i be the possible substitution for x_i to get s as an incomplete positive example. We have

$$n_0 + n_1|s_1| + n_2|s_2| + \cdots + n_k|s_k| = n.$$

We consider all possible $|s_i|$'s, $1 \le i \le k$, which satisfy the above equation. There are at most n^k of them. Each solution determines the positions of the substitution strings in s, as well as constant symbols in s.

For each of them, we first check whether the corresponding constant symbols of p and s are consistent. If it is so, we then check, for each x_i, whether its corresponding substrings in s are consistent, i.e., 0 and 1 does not appear at the same corresponding position. For example, 0?100 and 0?110 are not consistent and ?01?0 and ?0?10 are consistent. □

Now we show our result on *incomplete positive* examples.

Theorem 4.2 *The following problem is* NP*-complete: given a finite set S of incomplete positive examples and an integer t, determine whether there is a two-variable string pattern p of length $\geq t$ such that each $s \in S$ is an incomplete positive example for p.*

Proof: This problem is in *NP* since a nondeterministic Turing machine can guess a two-variable pattern p and, for each $s \in S$, guess a consistent example s' and verify that s' and s are consistent and $s' \in L(p)$ in polynomial time.

We reduce the 3SAT problem to this problem. Let $U = \{u_1, u_2, \ldots, u_n\}$ be the set of variables and $C = \{c_1, c_2, \ldots, c_m\}$ be an arbitrary instance of 3SAT. Without loss of generality, we assume each variable u_i, $1 \leq i \leq n$, or its negation $\overline{u}_i$, occurs at least once in some clause c_j, $1 \leq j \leq m$. First, we use two examples and the length bound to form a tableau for a truth assignment for C. Define

$$s_1 = 0\#\$^{n+1}\#\underbrace{??\cdots?}_{n}\#\$^{n+1}\#1$$

and

$$s_2 = 1\#\$^{n+1}\#\underbrace{??\cdots?}_{n}\#\$^{n+1}\#0.$$

With length bound $t = 3n + 8$ and examples s_1 and s_2, any 2-variable pattern p must have the structure:

$$p = x\#\$^{n+1}\#a_1a_2\cdots a_n\#\$^{n+1}\#y,$$

where x and y are variables and $a_i \in \{0, 1, x, y, \#, \$\}$.

Let

$$s_3 = \underbrace{00\cdots0}_{n+1}\#\$^{n+1}\#\underbrace{??\cdots?}_{n}\#\$^{n+1}\#\underbrace{00\cdots0}_{n+1},$$

then it can force every a_i, $1 \leq i \leq n$, be a constant. Thus, the center part $a_1 \cdots a_n$ of p may be viewed as a truth assignment for C. That is, $\tau_p(u_i) = \mathit{true}$ if and only if $a_i = 1$.

For each clause c_i, we construct a string s_i' such that if an assignment τ_p satisfies c_i then s_i' is an incomplete example for p. Thus, $\{c_1, \ldots c_m\}$ is satisfiable if and only if there is a pattern p for all s_i', $1 \leq i \leq m$. For each clause $c_i = \{l_{i_1}, l_{i_2}, l_{i_3}\}$, $l_{i_j} = u_{i_j}$ or $\overline{u}_{i_j}$, there are seven truth assignments on variables u_{i_1}, u_{i_2}, u_{i_3} which satisfy c_i (e.g., $x_1 \vee \overline{x}_2 \vee x_3$ can be satisfied by *ttt*, *ttf*, *tft*, *tff*, *ftt*, *fft* and *fff*, where t means *true* and f means *false*). Let them be

$$b_i[j,1]\,b_i[j,2]\,b_i[j,3],\ 1 \leq j \leq 7,$$

where $b_i[j,k] \in \{t, f\}$, $1 \leq j \leq 7$, $1 \leq k \leq 3$. For each c_i, we define 7 substrings $r_{i,j}$, each of length n (recall that $s(m)$ is the mth character of string s):

$$r_{i,j}(m) = \begin{cases} 1 & \text{if } b_i[j,k] = t, m = i_k, k = 1,2,3 \\ 0 & \text{if } b_i[j,k] = f, m = i_k, k = 1,2,3 \\ ? & \text{otherwise.} \end{cases}$$

For example, assume $n = 6$ and $c_2 = x_1 \vee \overline{x}_4 \vee \overline{x}_6$, then

$$\begin{aligned} r_{2,1} &= 1??0?0, \\ r_{2,2} &= 1??0?1, \\ r_{2,3} &= 1??1?0, \\ r_{2,4} &= 1??1?1, \\ r_{2,5} &= 0??0?0, \\ r_{2,6} &= 0??0?1, \\ r_{2,7} &= 0??0?0. \end{aligned}$$

We define $s_i' =$

$$0\#\$^{n+1}\#r_{i,1}\#\$^{n+1}\#r_{i,2}\#\$^{n+1}\#r_{i,3}\#\$^{n+1}\#r_{i,4}\#\$^{n+1}\#r_{i,5}\#\$^{n+1}\#r_{i,6}\#\$^{n+1}\#r_{i,7}\#\$^{n+1}\#0.$$

It can be checked that a pattern p matches s_i' only when $a_1 \cdots a_n$ matches with some $r_{i,j}$, $1 \leq j \leq 7$ (and x matches $0\#\cdots\#r_{i,j-1}$ and y matches $r_{i,j+1}\#\cdots\#0$). Thus sample $S = \{s_1, s_2, s_3\} \cup (\cup_{i=1}^{m}\{s_i'\})$ satisfies our requirement.

Note. The substring $\#\$^{n+1}\#$ of every example can be replaced by $10^{n+1}1$ without changing the proof. So, $\|\Sigma\| = 2$ is sufficient for this proof. □

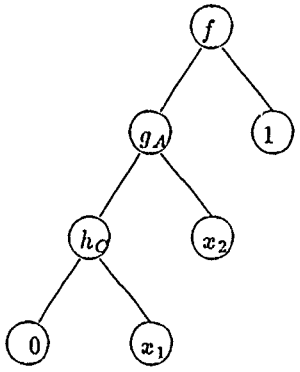
Through an easy extension of the above proof, we can show that the *NP*-completeness result holds for the cases $k \geq 3$.

Corollary 4.3 *For any $k \geq 2$, the following problem is* NP*-complete: given a finite set S of incomplete positive examples and an integer t, determine whether there is a k-variable string pattern p of length $\geq t$ such that each $s \in S$ is an incomplete example for p.*

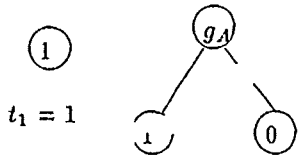
5 Tree Patterns

In this section, we generalize one-dimensional string patterns to two-dimensional tree patterns. Let $\mathcal{F}$ be a countable set disjoint from Σ and X. Elements in $\mathcal{F}$ are called *function symbols*. A *tree pattern* is a non-null rooted directed tree, where its vertices are labeled and edges are ordered. The internal vertices of a tree pattern are labeled by function symbols whose direct subtrees are their arguments. The external vertices (leaves) are labeled by constants or variables. The set $L(t)$ defined by tree pattern t is the set of all constant trees obtained by substituting non-null constant trees for variables occurring in t. For each tree t, we let $|t|$ be the number of vertices of tree pattern t. As an example, t in Figure 1 is a two-variable tree pattern. t_1 and t_2 are two constant trees. $t_3 = t[t_1/x_1, t_2/x_2]$ is an element in set $L(t)$ by substituting t_1 for x_1 and t_2 for x_2.

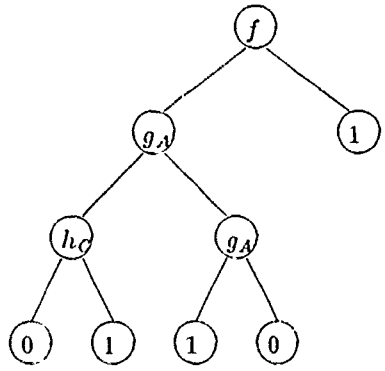
An internal vertex labeled by f is called *associative* if $f_A(a, f_A(b,c)) = f_A(f_A(a,b),c)$ and called *commutative* if $f_C(a,b) = f_C(b,a)$, where subscripts A and C of f denote the associativity and commutativity properties. A vertex could be associative, commutative, both, or neither, according to its associated function symbol. A subtree with associative vertices



$t = f(g_A(h_C(0, x_1), x_2), 1)$



$t_2 = g_A(1, 0)$



$t_3 = f(g_A(h_C(0, 1), g_A(1, 0)), 1)$

Figure 1. Constant Trees and Tree Patterns

$f_A(a, f_A(b, c))$ can be *flattened* to $f_A(a, b, c)$. A constant tree or tree pattern is *normalized* if every subtree is flattened, i.e., two associative vertices with the same function symbol have no direct parent-child relationship. For example, tree t in Figure 1 is normalized and tree t_3 is not. Tree t_3 can be flattened as $f(g_A(h_C(0, 1), 1, 0), 1)$. In the following, trees or tree patterns are all in the normalized form. String patterns can be viewed as depth-1 tree patterns with associativity property on the root.

5.1 Term Matching and Tree Patterns

The *term matching* (or membership) problem is, given a tree pattern t and a constant tree s, to decide whether $s \in L(t)$. Restrictions may be put on tree patterns t, when considering term matching. Some standard restrictions are the number of occurrences of each variable, the number of variables, and the properties of associativity and commutativity (e.g. AC term matching).

Term matching is one of fundamental problems in the area of term rewriting. The problem has been widely studied with respect to trees without associative or commutative properties, as well as one- and two-occurrence AC trees. The non-AC term matching problem has very efficient algorithm [DKM84] [DKS86]. The one-occurrence AC term matching problem has been proved polynomial-time solvable [BKN85] and the two-occurrence problem (even with only associativity or commutativity restriction) has been proved *NP*-complete [VR89]. Indeed from our comment about the relation between string patterns and associative tree patterns, the *NP*-completeness of the two-occurrence associative term matching problem also follows from a generalization of Angluin's *NP*-completeness result on the membership problem for string patterns (Proposition 2.2). We can also consider k-variable membership problems in string and tree patterns. For string patterns, the k-variable membership problem for any $k \geq 1$ is in NL (nondeterministic log-space), while the complexity of the k-variable AC term matching problem is still unknown for any $k \geq 1$. The less restrictive versions of the k-variable associative (or commutative) term matching problem is provable in NL for any $k \geq 1$. This further justifies that tree patterns with additional restrictions are a natural generalization of string patterns.

5.2 Positive Results

A non-associative, non-commutative tree is a tree such that all its internal vertices are labeled by non-associative, non-commutative function symbols. The positions of two subtrees of a vertex of a non-associative, non-commutative tree can not be exchanged because they are ordered. The *maximum tree pattern problem* is solvable in polynomial time with respect to this type of trees. We give, in Figure 2, a procedure MAX-TREE-PATTERN which takes as input a set of non-associative, non-commutative constant trees and outputs a maximum common tree pattern.

```
procedure MAX-TREE-PATTERN(T_1, T_2, ..., T_r)
begin
1.  t ← empty tree;
    (* Trace all trees T_i, 1 ≤ i ≤ r, in inorder
       simultaneously *)
2.  while there are unvisited vertices do
3.    Visit next unvisited vertex;
4.    if any two of current vertices
        are different
5.    then if current subtree of T_i, 1 ≤ i ≤ r, is
           equal to subtree pointed by p_{T_i,x_j}
6.         then set the corresponding vertex
             of t to x_j;
7.         else let x_l be a new variable and set
             pointer p_{T_i,x_l}, 1 ≤ i ≤ r, to the
             corresponding subtree rooted
             by current vertex of T_i;
8.           Set the corresponding vertex
             of t to x_l;
9.    else set the corresponding vertex of t to
        the symbol of current vertices;
10.   if the current vertex of t is set to a variable
11.   then mark all vertices in current subtrees as
        visited;
    end;
12. return(t);
end.
```

Figure 2: Finding Common Tree Patterns

We overlap all input trees $T_1, T_2, \cdots, T_r$ and substitute variables for unmatched corresponding subtrees. If two substituted subtrees are equal in every tree T_i, $1 \leq i \leq r$, then they are replaced by the same variable. This procedure will terminate since unvisited vertices decrease by at least one for every iteration of the while loop. The tree pattern t we get is the largest. If it is not, i.e., there exists a tree pattern t' such that $\cup_{i=1}^{r}\{T_i\} \subseteq L(t')$ and $|t'| > |t|$, then at least one *branch* b' (*path* from the root to a leaf) of tree pattern t' is longer than the corresponding branch b of tree pattern t. If the end (leaf) of branch b is a variable, then at least two T_i's corresponding positions are different. However, the branch b' is longer and so must have a function symbol at this position. Thus one of T_i's cannot be contained in $L(t')$. If the end of branch b is a constant, then all corresponding branches of T_i's are shorter than branch b'. All T_i's are not in $L(t')$. Thus, the size of tree pattern t is maximum for all T_i's. It is not too hard to verify that the procedure MAX-TREE-PATTERN runs in polynomial time.

Theorem 5.1 *There is a polynomial-time algorithm that finds the maximum tree pattern for a finite set of non-associative, non-commutative constant trees.*

For any fixed $k \geq 1$, we can also find the maximum k-variable tree pattern by first constructing the maximum tree pattern from the above procedure and then simply using exhaustive search to reduce the number of variables to $\leq k$.

Theorem 5.2 *For any $k \geq 1$, there is a polynomial-time algorithm to find the maximum k-variable tree pattern for a finite set of non-associative, non-commutative constant trees.*

5.3 Negative Results

For AC trees, the maximum tree pattern problem is *NP*-complete, even if the number of variables is fixed to two. However, the complexity of the one-variable case is still unknown.

Theorem 5.3 *The following problem is* NP*-complete: given a finite set S of associative, commutative constant trees and an integer r, determine whether there is a two-variable tree pattern t of size $\geq r$ such that $S \subseteq L(t)$.*

Proof: (Sketch) We reduce ONE-IN-THREE 3SAT with no negated literals [GJ79] to this problem. Let $U = \{u_1, u_2, \ldots, u_n\}$ be the set of variables and $C = \{c_1, c_2, \ldots, c_m\}$ be a set of clauses each containing three variables. We say that C is satisfiable if and only if there exists a truth assignment such that exact one variable of each clause is assigned to *true*. Without loss of generality, assume each variable u_i, $1 \leq i \leq n$, or its negation, occurs at least once in some c_j, $1 \leq j \leq m$. Let $r = 2n + 4$. For each clause $c_i = \{u_{i_1}, u_{i_2}, u_{i_3}\}$, $1 \leq i \leq m$, we generate

$$s_i = f_{AC}(t[i_1, i_2, i_3], t[i_2, i_3, i_1], t[i_3, i_1, i_2])$$

where $t[j,k,l]$ is a depth-1 tree with root h_{AC} and $2n - 3$ leaves: $\{g_m(0), g_m(1) : m \notin \{j,k,l\}\} \cup \{g_i(1), g_k(0), g_l(0)\}$. We also define

$$s_0 = f_{AC}(t_1, t_2, t_3),$$

where t_j, $j = 1, 2, 3$, is a depth-1 tree with a root h_{AC} and $2n + 1$ leaves: $\{g_m(0), g_m(1) : 1 \leq m \leq n\} \cup \{l_j(0)\}$.

Let $S = s_0 \cup (\cup_{i=1}^{m}\{s_i\})$. We want to prove that C is satisfiable by τ if and only if there is an associative, commutative tree pattern t of size $r = 2n+4$ such that $S \subseteq L(t)$. For the forward direction, we can see that tree pattern

$$f_{AC}(h_{AC}(g_1(a_1), \ldots, g_n(a_n), \mathrm{x}), \mathrm{y}),$$

is suitable, where x and y are variables and $a_i = 1$ if $\tau(u_i) = true$.

For the backward direction, a two-variable tree pattern t whose $L(t)$ contains S cannot be of the form.

$$f_{AC}(h_{AC}(\cdots), h_{AC}(\cdots), b)$$

for any variable b or a tree b rooted with h_{AC}; otherwise t must contain at least three variables because each subtree h_{AC} of s_0 has a different function symbol l_j. Therefore, the only suitable tree patterns of size $\geq 2n + 4$ are of the following form:

$$f_{AC}(h_{AC}(g_1(a_1), \ldots, g_n(a_n), \mathrm{x}), \mathrm{y}),$$

where $a_i \in \{0, 1\}$ for $1 \leq i \leq n$. Since each variable u_i appears at least once in some clause c_j, example s_j guarantees that only one subtree of h_{AC} is rooted by

g_i. We define $\tau(u_i) = true$ if the argument a_i of g_i is 1. □

The above proof can be extended to the cases $k \geq 3$.

Corollary 5.4 *For any $k \geq 2$, the following problem is NP-complete: given a finite set S of associative, commutative constant trees and an integer r, determine whether there is a k-variable tree pattern t of size $\geq r$ such that $S \subseteq L(t)$.*

6 Conclusions

We have investigated pattern learning problems in three new directions. Recently, we characterize the precise complexity of the problem FCP to be Σ_2^p-complete. Many problems mentioned in our paper remain open. We restate them here.

(1) For any fixed $k \geq 2$, find longest k-variable patterns for positive examples.

(2) For any fixed $k \geq 1$, find k-variable patterns for positive and negative examples.

(3) Find longest one-variable patterns for incomplete positive examples.

(4) Find one-variable AC tree patterns for positive and negative constant trees.

7 Acknowledgements

The authors would like to thank one of the program committee for his/her valuable comments.

References

[An80] D. Angluin, Finding patterns common to a set of strings, *J. of Computer and System Sciences* 21:46-62, 1980,

[AS83] D. Angluin, C. Smith, Inductive inference: theory and methods, *Computing Survey* 15:237-269, 1983.

[BEHW87] A. Blumer, A. Ehrenfeucht, D. Haussler, M. Warmuth, Occam's razor, *Information Processing Letters* 24:377-380, 1987.

[BEHW89] A. Blumer, A. Ehrenfeucht, D. Haussler, M. Warmuth, Learnability and the Vapnik-Chervonenkis dimension, *J. of ACM* 36(4):929-965, 1989.

[BKN85] D. Benanav, D. Kapur, P. Narendran, Complexity of matching problems, *Proc. 1st Conference on Rewriting Techniques and Applications*, Springer Verlag, 1985.

[DKM84] C. Dwork, P. Kanellakis, J.C. Mitchell, On the sequential nature of unification, *J. of Logic Programming* 1:35-50.

[DKS86] C. Dwork, P. Kanellakis, L. Stockmeyer, Parallel algorithm for term matching, *Proc. 8th Conference on Automated Deduction*, 1986.

[GJ79] M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[GS88] W.I. Gasarch, C.H. Smith, Learning via queries, *Proc. 29th IEEE Symposium on Foundations of Computer Science*, pp.130-137, 1988.

[HKLW88] D. Haussler, M. Kearns, N. Littlestone, M.k. Warmuth, Equivalence of models for polynomial learnability, *Proc. 1st Workshop on Computational Learning Theory*, pp.42-55, Morgan Kaufmann, San Mateo, CA, 1988.

[KH87] K. Ko, C. Hua, A note on the two-variable pattern-finding problem, *J. of Computer and System Sciences* 34(1):75-86, 1987.

[KP89] M. Kearns, L. Pitt, A polynomial-time algorithm for learning k-variable pattern languages from examples, *Proc. 2nd Workshop on Computational Learning Theory*, pp.57-70, Morgan Kaufmann, San Mateo, CA, 1989.

[Ma89] A. Marron, *Learning Pattern Languages from Examples and from Queries*, Ph.D. Thesis, University of Houston, 1989.

[Sc89] R. Schapire, Pattern languages are not learnable, manuscript, 1989.

[TK90] W. Tzeng, K. Ko, Finding common patterns is complete for the second level of the polynomial time hierarchy, manuscript, 1990.

[Va84] L.G. Valiant, A theory of the learnable, *Communication of ACM* 27(11):1134-1142, 1984.

[VR89] R.M. Verma, I.V. Ramakrishnan, Tight complexity bounds for some AC matching problems, *Symposium on Theoretical Aspects of Computer Science*, 1989.

Learning with Discrete Multi-Valued Neurons

Zoran Obradovic* and Ian Parberry
Department of Computer Science,
Penn State University,
University Park, Pa. 16802.

Abstract

Analog neural networks of limited precision are essentially k-ary neural networks. That is, their processors classify the input space into k regions using $k-1$ parallel hyperplanes by computing k-ary weighted multilinear threshold functions. The ability of k-ary neural networks to learn k-ary weighted multilinear threshold functions is examined. The well known perceptron learning algorithm is generalized to a k-ary perceptron algorithm with guaranteed convergence property. Littlestone's winnow algorithm is superior to the perceptron learning algorithm when the ratio of the sum of the weights to the threshold value of the function being learned is small. A k-ary winnow algorithm with a mistake bound which depends on this value and the ratio between the largest and smallest thresholds is presented.

1 Introduction

In (Obradovic & Parberry, 1990) it was shown that analog neural networks of limited precision are essentially k-ary neural networks (that is, their processors classify $\mathbf{R}^n$ into k regions using $k-1$ parallel hyperplanes) and their computing power was examined. Here, we investigate their learning power. One of the results from that reference was that there is no canonical set of threshold values for a k-ary perceptron when $k > 3$, although they exist for binary and ternary neural networks. This indicates that learning algorithms for k-ary neural networks which modify only the weights are not necessary convergent. Here we show that matters can be improved by learning both the thresholds and the weights. A preliminary version of the results from this paper appear in (Obradovic & Parberry, 1989).

The main body of this paper is divided into three sections. The first section sketches definitions of the k-ary neural network model and learning in that model. The second section contains a k-ary perceptron learning algorithm (derived from the binary perceptron learning algorithm) and its convergence proof. The third section contains a k-ary winnow algorithm (derived from Littlestone's winnow algorithm (Littlestone, 1987, 1989)) and its mistake bound proof.

*On leave from the Mathematical Institute, Belgrade, Yugoslavia.

2 A General Framework for Learning

Let $k \in \mathbf{N}$, and $\mathbf{Z}_k = \{0, \ldots, k-1\}$. A *$k$-ary neural architecture* is a k-ary neural network with the weights, thresholds and initial activation levels left unspecified. That is, it is a 4-tuple $A = (k, V, I, O)$, where:

$k \in \mathbf{N}$ is the number of logic levels,

V is a finite set of *processors*, or *gates*,

$I \subseteq V$ is a set of *input processors*,

$O \subseteq V$ is a set of *output processors*.

$A(a, w, h)$ denotes the *k-ary neural network* (k, V, I, O, a, w, h), where

$a : V - I \rightarrow \mathbf{Z}_k$ is a set of *initial activation levels*,

$w : V \times V \rightarrow \mathbf{R}$ is a *weight assignment*,

$h : V \rightarrow \mathbf{R}^{k-1}$ is a *threshold assignment*.

The processors of a k-ary neural network are relatively limited in computing power. Processor $v \in V$ has $k-1$ thresholds $h_1(v), \ldots, h_{k-1}(v)$, and if its weighted input sum is between $h_i(v)$ and $h_{i+1}(v)$ it has i for output.

More formally, a *k-ary function* is a function $f : \mathbf{Z}_k^n \rightarrow \mathbf{Z}_k$. Let F_k^n denote the set of all n-input k-ary functions. Define $\Theta_k^n : \mathbf{R}^{n+k-1} \rightarrow F_k^n$ by $\Theta_k^n(w_1, \ldots, w_n, h_1, \ldots, h_{k-1}) : \mathbf{R}_k^n \rightarrow \mathbf{Z}_k$, where

$$\Theta_k^n(w_1, \ldots, w_n, h_1, \ldots, h_{k-1})(x_1, \ldots, x_n) = i$$

$$\text{iff} \qquad h_i \leq \sum_{i=1}^{n} w_i x_i < h_{i+1}.$$

Here and throughout this paper, we will assume that $h_1 \leq h_2 \leq \ldots \leq h_{k-1}$, and for convenience define $h_0 = -\infty$ and $h_k = \infty$. The set of *k-ary weighted multilinear threshold functions* is the union, over all $n \in \mathbf{N}$, of the range of Θ_k^n. Each processor of a k-ary neural network can compute a k-ary weighted multilinear threshold function of its inputs.

Let $A = (A_1, A_2, \ldots)$ and $f = (f_1, f_2, \ldots)$ where $A_n = (k, V_n, I_n, O_n)$ is a neural architecture with $\|I\| = n$, and $f_n : \mathbf{R}^n \rightarrow \mathbf{Z}_k$. A *learning algorithm* for f on A is a relativized algorithm L with an oracle for f which on input n outputs a series of distinct initial activation, weight, and threshold assignments $\langle a_0, w_0, h_0\rangle$, $\langle a_1, w_1, h_1\rangle, \ldots, \langle a_t, w_t, h_t\rangle$ such that the neural network $M_n = A_n(a_t, w_t, h_t)$ computes f_n. We will consider learning algorithms for k-ary weighted multilinear threshold functions on *neural circuits* (that is, layered neural networks without feedback).

Resources of interest include those of M_n, and those of L. The former include the size (number of processors), depth (number of layers), and weight (sum of all the weights) of the circuit. The latter include the latency and the mistake bound, defined as follows. The *latency* of learning algorithm is the worst case running time between the output of one set of assignments and the next. We will measure *unit-cost* latency, that is, we will assume that L is implemented on a digital computer with word-size large enough that each elementary arithmetic and logic operation can be implemented in constant time. The *mistake bound* is the worst case total number of distinct assignments output.

If f is a k-ary weighted multilinear threshold function, we say that $(w_1, \ldots, w_n, t_1, \ldots, t_{k-1}) \in \mathbf{R}^{n+k-1}$ is a *representation* of f iff

$$f = \Theta_k^n(w_1, \ldots, w_n, t_1, \ldots, t_{k-1}).$$

Note that each k-ary weighted multilinear threshold function has many representations. We will consider the problem of learning k-ary weighted multilinear threshold functions, and for the most part be concerned with learning them on the minimal architecture consisting of a single k-ary processor. That is, we will be learning a representation for a k-ary weighted multilinear threshold function f, given only an oracle for f. All of our learning algorithms will be expressed in a high-level pseudocode. Initial activation levels will always be zero.

We will consider two new resources, called the *height* and *width* of a representation, which give some indication of the relationships between the weights and the thresholds, and between the thresholds themselves (respectively), to be defined later.

3 A k-ary Perceptron Learning Rule

The *perceptron learning problem* is the problem of learning binary weighted linear threshold functions on a binary neural network consisting of a single processor (called a *perceptron* for historical reasons). There is a well-known algorithm for the perceptron learning problem which uses the so-called *perceptron learning rule* to derive successive weights. The algorithm is described in Figure 1.

Theorem 3.1 (The Perceptron Convergence Theorem) *The perceptron learning algorithm for learning n-input binary weighted linear threshold functions described in Figure 1 terminates.*

```
procedure perceptron(n)
  for i := 1 to n do w_i := 0 ;
  repeat
    for each x = (x_1, ..., x_n) ∈ Z_2^n do
      p := Θ_2^n(w_1, ..., w_n, 0)(x);
      if f(x) ≠ p
        then perceptron_update(x, p);
      Output ⟨w_1, ..., w_n⟩
  until f(x) = Θ_2^n(w_1, ..., w_n, 0)(x)
        for all x ∈ Z_2^n.

procedure perceptron_update(x, p)
  if f(x) > p then sign := 1
    else sign := −1;
  for i := 1 to n do w_i := w_i + sign * x_i;
```

Figure 1: The Perceptron Learning Algorithm.

Proof: See, for example, (Duda & Hart, 1973), (Minsky & Papert, 1969), (Nilsson, 1962) or (Novikoff, 1962). □

The initial weights can be set to any value in the for-loop on line 1 in Figure 1. The members of $\mathbf{Z}_2^n$ can be used in any order in the for-loop on line 3, provided every member is used an infinite number of times in the algorithm. Also, the value added to w_i in the for-loop of *perceptron_update* procedure can be multiplied by some constant $c_j \in \mathbf{R}^+$ at the j^{th} call of that procedure provided

$$\lim_{m\rightarrow\infty} \sum_{i=1}^{m} c_i = \infty$$

and

$$\lim_{m\rightarrow\infty} \frac{\sum_{i=1}^{m} c_i^2}{(\sum_{i=1}^{m} c_i)^2} = 0.$$

Furthermore, the algorithm will learn any weighted linear threshold function whose domain is some finite subset of $\mathbf{Z}_2^n$. We will make use of this fact later. The latency of the algorithm is clearly linear in n. The worst-case mistake bound appears no better than exponential in n.

The minimal architecture for learning n-input k-ary weighted multilinear threshold functions is a single k-ary weighted multilinear threshold gate with n inputs, which we will call a *k-ary perceptron*. It was shown in (Obradovic & Parberry, 1990) that there is no canonical set of threshold values for a k-ary perceptron when $k > 3$. This suggests that the thresholds must be learned in addition to the weights.

Even if the threshold values are known in advance, many obvious extensions to the perceptron learning rule (such as that shown in Figure 2) which modify only the weights do not necessary terminate for all

```
procedure thresholdperceptron(n, k, t_1, ..., t_{k-1})
  for i := 1 to n do w_i := 0;
  repeat
    for each x ∈ Z_k^n do
      p := Θ_k^n(w_1, ..., w_n, t_1, ..., t_{k-1})(x)
      if f(x) ≠ p
        then thresholdperceptron_update(x, p);
      Output ⟨w_1, ..., w_n⟩
  until f(x) = Θ_k^n(w_1, ..., w_n, t_1, ..., t_{k-1})(x)
        for all x ∈ Z_k^n.

procedure thresholdperceptron_update(x, p)
  if f(x) > p then sign := 1
    else sign := −1;
  for i := 1 to n do w_i := w_i + sign * x_i;
```

Figure 2: A Trial k-ary Perceptron Learning Algorithm for Known Thresholds.

choices of ordering of sample inputs in line 4. For example, suppose $k = 2$ and $n = 2$ (similar examples can be found for arbitrary n and k using the same principles). Consider $f = \Theta_2^2(4,3,7,8)$. Suppose we use the algorithm described in Figure 2 to find weights w_1, w_2 such that $\Theta_2^2(w_1, w_2, 7, 8) = f$. After considering points $(2,0)$ and $(2,2)$, we have weights $(w_1, w_2) = (4,2)$. All points are correctly classified using these weights except for the point $(1,1)$. Thus there is no change to the weights until point $(1,1)$ is considered, at which time the new weights become $(5,3)$. Once again, all points are correctly classified using these weights except for the point $(1,1)$. Thus there is no change to the weights until point $(1,1)$ is reconsidered, at which time the new weights are again $(4,2)$. Thus the weights cycle between $(4,2)$ and $(5,3)$ without ever reaching an acceptable solution. Matters are not improved by making obvious changes to Figure 2, for example, instead of adding a multiple of x_i in the for loop of *thresholdperceptron_update* procedure, substituting one if $x_i > 0$ and zero otherwise.

However, matters can be improved by learning both the thresholds and the weights.

It can be shown from first principles that the k-ary perceptron learning algorithm described in Figure 3 terminates. Termination can more easily be proved as a corollary of the Perceptron Convergence Theorem as follows.

Definition. If f is an n-input k-ary weighted multilinear threshold function, the *orthogonal slice function* for f is a binary weighted linear threshold function g such that for all $x \in \mathbf{Z}_k^n$ and all $i \in \mathbf{Z}_k$,

$$f(x) \geq i \Leftrightarrow g(x, y(i)) = 1,$$

where $y : \{1, \ldots, k-1\} \rightarrow \{0,1\}^{k-1}$ is defined by

$$y(i) = (y_1, \ldots, y_{k-1}) \text{ with } y_j = 1 \text{ iff } j = i.$$

```
procedure multiperceptron(n, k)
  for i := 1 to n do w_i := 0;
  for i := 1 to k − 1 do t_i := 0;
  repeat
    for each x = (x_1, ..., x_n) ∈ Z_k^n do
      p := Θ_k^n(w_1, ..., w_n, t_1, ..., t_{k-1})(x);
      if f(x) ≠ p
        then multiperceptron_update(x, p);
      Output ⟨w_1, ..., w_n, t_1, ..., t_{k-1}⟩
  until f(x) = Θ_k^n(w_1, ..., w_n, t_1, ..., t_{k-1})(x)
        for all x ∈ Z_k^n.

procedure multiperceptron_update(x, p)
  if f(x) > p
    then t_p + 1 := t_p + 1 − 1; sign := 1
    else t_p := t_p + 1; sign := −1;
  for i := 1 to n do w_i := w_i + sign * x_i;
```

Figure 3: The k-ary Perceptron Learning Algorithm.

Lemma 3.2 $f = \Theta_k^n(w_1, \ldots, w_n, t_1, \ldots, t_{k-1})$ *iff* $\Theta_2^{n+k-1}(w_1, \ldots, w_n, -t_1, \ldots, -t_{k-1}, 0)$ *is the orthogonal slice function for* f.

Proof: Follows immediately from the definition of the orthogonal slice function □

Theorem 3.3 (The k-ary Perceptron Convergence Theorem) *The k-ary perceptron learning algorithm for learning n-input k-ary weighted multilinear threshold functions described in Figure 3 terminates.*

Proof: (Sketch) The learning algorithm, instead of learning f, learns the orthogonal slice function for f using the binary perceptron learning algorithm shown in Figure 1. The orthogonal slice function for f is guaranteed to exist by (the "only-if" part of) Lemma 3.2. Once the orthogonal slice function has been learned, f can be reconstructed using (the "if" part of) Lemma 3.2. The algorithm is guaranteed to terminate by the Perceptron Convergence Theorem. It is clear that the algorithm realizes Figure 3. □

The latency of the k-ary perceptron learning algorithm is $O(n)$, and the mistake bound is no worse than for the binary perceptron learning algorithm on $n + k$ inputs.

A second candidate architecture for learning k-ary weighted multilinear threshold functions consists of $\lceil \log k \rceil$ binary perceptrons, the i^{th} of which learns the i^{th} bit of the output value, together with a single k-ary weighted multilinear threshold gates with exponentially increasing weights which converts the binary output of these gates into the corresponding member of $\mathbf{Z}_k$. Unfortunately this cannot work because the last binary perceptron is expected to learn the least significant bit of the k-ary output, which is not necessarily

```
procedure netperceptron(n, k)
  for i := 1 to k − 1 do
    for j := 1 to n do w_{i,j} := 0;
  repeat
    for each x = (x_1, ..., x_n) ∈ Z_k^n do
      for each i ∈ Z_k do neuron_update(x, i)
    Output ⟨w_{1,1}, ..., w_{k−1,n}⟩
  until ( f(x) ≥ i) ⇔ (Θ_2^n(w_{i,1}, ..., w_{i,n}, 0)(x) = 1)
        for all x ∈ Z_k^n and 1 ≤ i ≤ k − 1.

procedure neuron_update(x, i)
  p := Θ_2^n(w_{i,1}, ..., w_{i,n}, 0)(x);
  if (f(x) ≥ i) and (p = 0) then sign := 1
    else if (f(x) < i) and (p = 1)
        then sign := −1
        else sign := 0;
  for j := 1 to n do w_{i,j} := w_{i,j} + sign * x_i;
```

Figure 4: The k-ary Perceptron Learning Algorithm for a Depth 2 Circuit.

a binary weighted threshold function.

A third candidate architecture for learning k-ary weighted multilinear threshold functions consists of a depth 2 circuit of size k. The first layer consists of $k-1$ binary perceptrons, each connected to all of the inputs. The second layer consists of a single k-ary perceptron, connected to all of the gates in the first layer. The thresholds of the first layer are all zero. The thresholds of the k-ary perceptron are $1, 2, \ldots, k-1$. The weights of the connections from the first layer to the second are all one. The weights of the connections from the inputs to the first layer will be learned.

Let $w_{i,j}$ denote weight from the j^{th} input to the i^{th} gate on the first layer, where $1 \leq i \leq k-1$ and $1 \leq j \leq n$. Suppose we are to learn a k-ary weighted multilinear threshold function f. The first level essentially computes the orthogonal slice function for f, and the second level converts this to a value from $\mathbf{Z}_k$. More precisely, the i^{th} gate on the first level, $1 \leq i \leq k-1$, will output one on input x iff $f(x) \geq i$. This implies that exactly $f(x)$ of the gates in the first layer will be active. The output gate sums the number of active gates in the first layer.

It is clear that by performing the binary perceptron learning rule in parallel for all $k-1$ gates in the first layer, the network will learn arbitrary k-ary weighted multilinear threshold functions. The learning algorithm is described in Figure 4. It has a latency of $O(nk)$. Its mistake bound may be better than that of Figure 3 in practice since it learns arbitrary separating hyperplanes, rather than parallel ones. However, the worst case mistake bound remains apparently exponential.

4 A k-ary Winnow Algorithm

A representation $(w_1, \ldots, w_n, t_1, \ldots, t_{k-1}) \in \mathbf{R}^{n+k-1}$ of a k-ary weighted multilinear threshold function is *positive* iff $w_i \geq 0$ for all $1 \leq i \leq n$. A k-ary weighted multilinear threshold function is *positive* iff it has a positive representation.

A positive representation $(w_1, \ldots, w_n, t_1, \ldots, t_{k-1})$ has *separation* $\lambda \in \mathbf{R}^+$, $0 < \lambda \leq 1$, if for all $x = (x_1, \ldots, x_n) \in \mathbf{Z}_k^n$, and all $i \in \mathbf{Z}_k$, $i < k-1$,

$$f(x) \leq i \text{ iff } \sum_{j=1}^{n} w_j x_j \leq (1-\lambda) t_{i+1}.$$

The *width* of a representation

$$(w_1, \ldots, w_n, t_1, \ldots, t_{k-1})$$

is the ratio t_{k-1}/t_1. Note that all representations of a binary weighted linear threshold function have width one. A k-ary weighted multilinear threshold function has *width* (at most) d iff it has a representation of width d.

The *height* of a representation

$$(w_1, \ldots, w_n, t_1, \ldots, t_{k-1})$$

of width d is the ratio

$$\sum_{i=1}^{n} \frac{|w_i|}{t_{k-1}} + 1 - \frac{1}{d}.$$

A k-ary weighted multilinear threshold function has *height* (at most) h iff it has a representation of height h. A k-ary weighted multilinear threshold function is (λ, h, d)-*separable* if it has a positive representation of separation $\lambda > 0$, height h, and width d. Since all binary weighted linear threshold functions have width 1, we will write (λ, h)-separable when $k = 2$.

When learning weighted linear threshold functions, we can without loss of generality restrict ourselves to learning positive ones. If we need to learn a weighted linear threshold function with negative weights, we can substitute a positive function of the same height, and perform a minimal amount of pre-processing of the inputs:

Lemma 4.1 *For each representation $(v_1, \ldots, v_n, t)$ of height h there exists a positive representation $(w_1, \ldots, w_n, r)$ of height h and a function $g : \mathbf{Z}_2^n \to \mathbf{Z}_2^n$ of the form $g(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ where for $1 \leq i \leq n$, , either $y_i = x_i$ or $y_i = \overline{x}_i$, such that for all $x \in \mathbf{Z}_2^n$,*

$$f(x) = \Theta_2^n(w_1, \ldots, w_n, r)(g(x))$$

Proof: We make use of an elementary technique due to (Muroga, 1971) (see also Theorem 4.5.2 of (Parberry, 1990)). Suppose

$$f(x) = \Theta_2^n(v_1, \ldots, v_n, t)$$

Then

$$w_i = |v_i| \text{ for } 1 \leq i \leq n,$$

and

$$r = t + \sum_{i=1}^{n} (|\, v_i \,| - v_i)/2,$$

and $y_i = 0.5 + (x_i - 0.5)v_i/|\, v_i \,|$. The new representation has height at most h since its denominator is larger than that of the original representation, whilst its numerator is the same. □

The threshold value in a positive representation is not important.

Lemma 4.2 *For each positive representation* $(v_1, \ldots, v_n, t)$ *of height* h *and separation* λ *with* $t > 0$, *and all* $r \in \mathbf{R}^+$, *there is a positive representation* $(w_1, \ldots, w_n, r)$ *of height* h *and separation* λ *such that*

$$\Theta_2^n(v_1, \ldots, v_n, t) = \Theta_2^n(w_1, \ldots, w_n, r).$$

Proof: Set $w_i = v_i r/t$ for $1 \leq i \leq n$. □

Littlestone (1987, 1989) proposed a learning algorithm, called the *winnow algorithm* (see Figure 5) for learning n-input binary, positive, (λ, h)-separable functions on a single n-input perceptron. The algorithm takes as parameter a constant α, and learns a positive representation with threshold value n (such a representation exists, by Lemma 4.2). The latency of the binary winnow algorithm is clearly linear in n. The mistake bound is given by the following theorem.

Theorem 4.3 *If* $\alpha = 0.5\lambda + 1$, *then the number of mistakes made by the winnow algorithm in Figure 5 learning an* n*-input, positive,* (λ, h)*-separable weighted linear threshold function is at most*

$$\left(\frac{14 \log n}{\lambda^2} + \frac{5}{\lambda}\right) h + \frac{8}{\lambda^2}.$$

Proof: See (Littlestone, 1987). □

Later we will use the fact (Littlestone, 1989) that the winnow algorithm will learn any n-input, positive, (λ, h)-separable weighted linear threshold function whose domain is some finite subset of $\{\{0\} \bigcup [\delta, 1]\}^n$. In that case, the mistake bound from Theorem 4.1 depends on $\log(n/\delta)$ instead of $\log n$.

The mistake bound is a significant improvement over the binary perceptron learning algorithm for weighted linear threshold functions with large separation and small height. In contrast, the best known mistake upper bound for the perceptron learning algorithm (see, for example, Duda & Hart, 1973) is polynomial in the weight of the best representation (if it is sufficiently large). It is known (see, for example, Muroga, 1971; Parberry, 1990) that there are weighted linear threshold functions for which the weight of the best representation is at least exponential in n, and it can be deduced that there are functions with exponential weight, polynomial height and inverse-polynomial separation. Whilst the perceptron learning algorithm appears to make exponentially many mistakes for these

```
procedure winnow(n, α)
  for i := 1 to n do w_i := 1;
  repeat
    for each x = (x_1, ..., x_n) ∈ Z_2^n do
      p := Θ_2^n(w_1, ..., w_n, n)(x);
      if f(x) ≠ p
        then winnow_update(x, p, α);
      Output ⟨w_1, ..., w_n, n⟩
  until f(x) = Θ_2^n(w_1, ..., w_n, t)(x)
        for all x ∈ Z_2^n.

procedure winnow_update(x, p, α)
  if f(x) > p then δ := α
    else δ := 1/α;
  for i := 1 to n do w_i := w_i δ^{x_i};
```

Figure 5: The Winnow Learning Algorithm.

functions, the winnow learning algorithm makes only polynomially many mistakes. If λ and h are constant, only $O(\log n)$ mistakes are made.

In the light of Lemma 4.1, the definition of (λ, h)-separability can be extended to non-positive weighted linear threshold functions as follows. A representation $(w_1, \ldots, w_n, t)$ has *separation* $\lambda \in \mathbf{R}^+$, $0 \leq \lambda \leq 1$, if for all $x = (x_1, \ldots, x_n) \in \mathbf{Z}_2^n$,

$$f(x) \leq 0 \text{ iff}$$

$$\sum_{j=1}^{n} |\, w_j \,|\, x_j \leq (1 - \lambda)(t + \sum_{j=1}^{n} (|\, w_j \,| - w_j)/2).$$

Hence we have:

Corollary 4.4 *Any* n*-input* (λ, h)*-separable weighted linear threshold function can be learned on a neural circuit of depth 2 and size at most* n *with latency* $O(n)$ *and mistake bound*

$$\left(\frac{14 \log n}{\lambda^2} + \frac{5}{\lambda}\right) h + \frac{8}{\lambda^2}.$$

Proof: The result is an immediate consequence of Lemma 4.1 and Theorem 4.3. □

We extended the winnow algorithm to k-ary weighted multilinear threshold functions. The *k-ary winnow algorithm* for learning n input, k-ary, positive, (λ, h, d) separable function on a single k-ary perceptron with n inputs is described in Figure 6.

The latency of the algorithm is $O(n + k)$. To prove the mistake bound we will use a new slice function which preserves positiveness.

Definition. If f is an n-input k-ary weighted multilinear threshold function, the *unary slice function* for f is a binary weighted linear threshold function g such that for all $x \in \mathbf{Z}_k^n$ and all $i \in \mathbf{Z}_k$,

$$f(x) \geq i \Leftrightarrow g(x, y(i)) = 1,$$

procedure *multiwinnow_learning*(n, k, α)
  for $i := 1$ **to** $n + k - 2$ **do** $w_i := 1$;
  $t_{k-1} = (k-1)(n+k-2)$;
  for $i := k-2$ **downto** 1 **do** $t_i = t_{i+1} - 1$;
  repeat
    for *each* $x = (x_1, \ldots, x_n) \in \mathbf{Z}_k^n$ **do**
      $p := \Theta_k^n(w_1, \ldots, w_n, t_1, t_2, \ldots, t_{k-1})(x)$;
      if $f(x) \neq p$ **then**
        multiwinnow_update$(x, p, \alpha^{1/(k-1)})$;
      Output $\langle w_1, \ldots, w_n, t_1, t_2, \ldots, t_{k-1} \rangle$.
  until $f(x) = \Theta_k^n(w_1, \ldots, w_n, t_1, \ldots, t_{k-1})(x)$
      for all $x \in \mathbf{Z}_k^n$

procedure *multiwinnow_update*(x, p, α)
  for $i := 1$ **to** n **do** $z_i := x_i$;
  for $i := 1$ **to** $k-2$ **do** $z_{n+i} := 0$;
  if $f(x) > p$ **then** $\delta := \alpha$; $ind := p + 1$
    else $\delta := 1/\alpha$; $ind := p$;
  for $i :=$ *ind* **to** $k-2$ **do** $z_{n+i} := 1$;
  for $i := 1$ **to** $n + k - 2$ **do** $w_i := w_i \delta^{z_i}$;
  for $i := k-2$ **downto** 1 **do** $t_i = t_{i+1} - w_{n+i}$;

Figure 6: The k-ary Winnow Learning Algorithm.

where $y : \{1, \ldots, k-1\} \rightarrow \{0,1\}^{k-1}$ is defined by

$$y(i) = (y_1, \ldots, y_{k-2}) \text{ with } y_j = 1 \text{ iff } j \geq i.$$

Lemma 4.5 *If* $(v_1, \ldots, v_n, t_1, \ldots, t_{k-1})$ *is a positive representation of height* h*, width* d*, and separation* λ *of a* k*-ary weighted multilinear threshold function* f*, then* $(w_1, \ldots, w_n, t_2 - t_1, t_3 - t_2, \ldots, t_{k-1} - t_{k-2}, t_{k-1})$ *is a positive representation of height* h *and separation* λ/d *of the unary slice function for* f.

Proof: Follows immediately from the definition of the unary slice function. □

Theorem 4.6 *If* $\alpha = 1 + \lambda/(2d)$*, then the number of mistakes made by the* k*-ary winnow algorithm in Figure 6 learning an* n*-input, positive,* (λ, h, d)*-separable* k*-ary weighted multilinear threshold function is bounded above by*

$$\left(\frac{14d^2 \log((k-1)(n+k-2))}{\lambda^2} + \frac{5d}{\lambda}\right)(k-1)h + \frac{8d^2}{\lambda^2}.$$

Proof: (Sketch) By Lemma 4.5 the learning algorithm, instead of learning a positive representation

$$(w_1, \ldots, w_n, t_1, t_2, , \ldots, t_{k-1})$$

of height h and separation λ for a k-ary weighted multilinear threshold function f , can learn a positive representation

$$(w_1, \ldots, w_n, t_2 - t_1, t_3 - t_2, , \ldots, t_{k-1} - t_{k-2}, t_{k-1})$$

of height h and separation $\lambda^* = \lambda/d$ of the unary slice function for f. Since inputs $(x_1, \ldots, x_n)$ for function f are from $\mathbf{Z}_k^n$, we can not apply the binary winnow learning algorithm directly to learn the unary slice function for f on inputs $(x, y(i)) = (x_1, \ldots, x_n, y_1, \ldots, y_{k-1})$. But, we can use the binary winnow learning algorithm with learning parameter α and threshold $t = n + k - 2$ to learn a modified unary slice function

$$(w_1, \ldots, w_n, t_2 - t_1, t_3 - t_2, , \ldots, t_{k-1} - t_{k-2}, t_{k-1}/(k-1))$$

on compressed inputs

$$(x_1/(k-1), \ldots, x_n/(k-1), y_1/(k-1), \ldots, y_{k-2}/(k-1))$$

from $\{\{0\} \bigcup [1/(k-1), 1]\}^n$.

Finally, observe that

$$\alpha^{z_i/k-1} w_i = (\alpha^{1/(k-1)})^{z_i} w_i$$

and also

$$\sum_{i=1}^{n} w_i \frac{x_i}{k-1} + \sum_{i=1}^{k-2} w_{n+i} \frac{y_i}{k-1} < (n + k - 2)$$

iff

$$\sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{k-2} w_{n+i} y_i < (k-1)(n+k-2).$$

So, for learning we can actually use binary winnow algorithm with learning parameter $\alpha^{1/(k-1)}$ and threshold $t = (k-1)(n+k-2)$ on the original inputs $(x_1, \ldots, x_n, y_1, \ldots, y_{k-1})$. It is easy to see that this algorithm realizes Figure 6. Substituting $\alpha = 1 + \lambda^*/2$, $t = n + k - 2$ and $\delta = 1/(k-1)$ in the Theorem 4.3 we obtain the mistake bound from the claim of this theorem. □

The mistake bound of the k-ary winnow algorithm is a significant improvement over the k-ary perceptron learning algorithm for (λ, h, d)-separable functions when λ is large and h and d are small.

A slightly better mistake bound can be obtained if the input $x \in \mathbf{Z}_k^n$ is encoded in binary as $x^* \in \mathbf{Z}_2^{n \log k}$ and the k-ary winnow algorithm is used on a k-ary perceptron with $n \log k$ binary inputs instead of n k-ary inputs. Instead of learning k-ary weighted multilinear threshold functions, we can substitute *binary-to-k-ary weighted multilinear threshold functions,* which are simply k-ary weighted multilinear threshold functions whose domain is restricted to $\mathbf{Z}_2^n$, and perform a small amount of pre-processing of the inputs. Without loss of generality, we will henceforth assume that k is a power of two. Define function $Encode : \mathbf{Z}_k^n \rightarrow \mathbf{Z}_2^{n \log k}$, which simply encodes a k-ary input in binary, by $Encode(x) = (y_1, \ldots, y_{n \log k})$ where $x = (x_1, \ldots, x_n) \in \mathbf{Z}_k^n$ and

$$x_i = \sum_{j=1}^{\log k} 2^{j-1} y_{j+(i-1)\log k}.$$

Lemma 4.7 *For every n-input, positive, k-ary weighted multilinear threshold function f of height h and depth d there exists an $(n \log k)$-input binary-to-k-ary weighted multilinear threshold function g of height $(k-1)h$ and width d such that for all $x \in \mathbf{Z}_k^n$, $f(x) = g(Encode(x))$.*

Proof: If $f = \Theta_k^n(w_1, \ldots, w_n, t_1, \ldots, t_{k-1}$ then g is the function $\Theta_k^n(w_{1,1}, \ldots, w_{n,\log k}, t_1, \ldots, t_{k-1}$ with domain restricted to $\mathbf{Z}_2^{n \log k}$, where $w_{i,j} = 2^{j-1}w_i$ for $1 \leq i \leq n$, $i \leq j \leq \log k$. □

Lemma 4.8 *If $(w_1, \ldots, w_{n+k-2}, t)$ is the representation of the unary slice function of a binary-to-k-ary weighted multilinear threshold function f, then $(w_1, \ldots, w_n, t_1, , \ldots, t_{k-1})$ is a representation of f, where*

$$t_i = t - \sum_{j=n+i}^{n+k-2} w_j.$$

Proof: Follows immediately from the definition of the unary slice function. □

Theorem 4.9 *Any n-input, positive, (λ, h, d)-separable binary-to-k-ary weighted multilinear threshold function can be learned on a k-ary perceptron with latency $O(n+k)$ and mistake bound*

$$\left(\frac{14d^2 \log(n+k-2)}{\lambda^2} + \frac{5d}{\lambda}\right) h + \frac{8d^2}{\lambda^2}.$$

Proof: (Sketch) The domain of binary-to-k-ary weighted multilinear threshold function is restricted to $\mathbf{Z}_2^n$. So, the learning algorithm, instead of learning f, can learn the unary slice function for f using the binary winnow learning algorithm. The threshold is chosen equal to $n+k-2$ by Lemma 4.2. The unary slice function for f is guaranteed to exist by Lemma 4.5. Once the unary slice function has been learned, f can be reconstructed using Lemma 4.8. The mistake bound is given by Theorem 4.3. □

Theorem 4.10 *Any n-input, positive, (λ, h, d)-separable k-ary weighted multilinear threshold function can be learned on a k-ary neural circuit of depth 4 and size $O(nk)$ with latency $O(n \log k + k)$ and mistake bound*

$$\left(\frac{14d^2 \log(n \log k + k - 2)}{\lambda^2} + \frac{5d}{\lambda}\right)(k-1)h + \frac{8d^2}{\lambda^2}.$$

Proof: (Sketch) The result is an immediate consequence of Lemma 4.7 and Theorem 4.9. A depth 3, size $O(nk)$ k-ary threshold circuit can compute function *Encode*. The details are left to the interested reader. □

The definition of (λ, h, d)-separability can be extended to non-positive weighted multilinear threshold functions as follows. A k-ary weighted multilinear threshold function $f : \mathbf{Z}_k^n \rightarrow \mathbf{Z}_k$ is (λ, h, d)-*separable* iff its binary encoded equivalent $f^* : \mathbf{Z}_2^{n \log k} \rightarrow \mathbf{Z}_k$ is $(\lambda, (k-1)h, d)$-separable, where separation is $\lambda \in \mathbf{R}^+$, $0 < \lambda \leq 1$, if for all $x \in \mathbf{Z}_2^{n \log k}$ and all $i \in \mathbf{Z}_k$, $i < k-1$,

$$f^*(x) \leq i \text{ iff}$$

$$\sum_{j=1}^{n \log k} |w_j| x_j \leq (1-\lambda)(t_{i+1} + \sum_{j=1}^{n \log k} (|w_i| - w_i)/2)).$$

Then we have the extension of the result to the non-positive case:

Corollary 4.11 *Any n-input (λ, h, d)-separable k-ary weighted multilinear threshold function can be learned on a k-ary neural circuit of depth 4 and size $O(nk)$ with latency $O(n \log k + k)$ and mistake bound*

$$\left(\frac{14d^2 \log(n \log k + k - 2)}{\lambda^2} + \frac{5d}{\lambda}\right)(k-1)h + \frac{8d^2}{\lambda^2}.$$

Proof: (Sketch) Suppose f is a (λ, h, d)-separable k-ary weighted multilinear threshold function. Then it has an $(n \log k)$-input $(\lambda, (k-1)h, d)$-separable binary-to-k-ary weighted multilinear threshold function f_1 by Lemma 4.7. By Lemma 4.5, f_1 has an $(n \log k + k - 2)$-input $(\lambda/d, (k-1)h)$-separable unary slice function f_2, which can be replaced by an $(n \log k + k - 2)$-input positive $(\lambda/d, (k-1)h)$-separable unary slice function f_3 by Lemma 4.1. A depth 2, size $O(nk)$ k-ary threshold circuit can compute function Encode and the negations of the appropriate inputs. The winnow algorithm is used to learn a representation for f_3. By Theorem 4.3, the latency is $O(n \log k + k)$ and the mistake bound is

$$\left(\frac{14d^2 \log(n \log k + k - 2)}{\lambda^2} + \frac{5d}{\lambda}\right)(k-1)h + \frac{8d^2}{\lambda^2}.$$

A careful analysis gives the required mistake bound. □

The k-ary winnow algorithm can also be used to learn k-ary weighted multilinear threshold functions on a network of size k and depth 2 by using essentially the same techniques as were used for the perceptron learning algorithm in Section 3. The details are left for the interested reader.

5 Conclusion

The study of k-ary neural networks was justified by the observation that they are closely related to analog neural networks of bounded precision. We have seen two learning results for k-ary neural networks. Firstly, we have demonstrated a k-ary perceptron learning rule with guaranteed convergence. Secondly, Littlestone's winnow algorithm, which learns binary weighted linear threshold functions with a mistake bound dependent on their height has been extended to a k-ary winnow algorithm whose mistake bound depends on the height and width of the k-ary weighted multilinear threshold function being learned.

5.0.1 Acknowledgements

The financial support of the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under grant numbers AFOSR 87-0400 and AFOSR 89-0168 and NSF grant CCR-8801659 to Ian Parberry is gratefully acknowledged.

5.0.2 References

Duda, R.O., and Hart, P.E., (1973) *Pattern classification and scene analysis,* John Willey, New York.

Littlestone, N., (1987) "Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm," *Machine Learning,* vol. 1, no. 2, pp.285-318.

Littlestone, N., (1989) "Mistake bounds and logarithmic linear-threshold learning algorithms," *Technical Report CRL-89-11,* University of California at Santa Cruz.

Minsky, M., and Papert, S., (1969) *Perceptrons,* MIT Press.

Muroga, S., (1971) *Threshold Logic and its Applications,* Wiley Interscience, New York.

Nilsson, N.J., (1962) *Learning Machines,* McGraw-Hill, New York, NY.

Novikoff, A., (1962) "On convergence proofs for perceptrons," *Proc. Symposium on Mathematical Theory of Automata,* New York, NY, pp. 615-622.

Obradovic, Z., and Parberry, I., (To Appear in 1990) "Analog neural networks of limited precision I: Computing with multilinear threshold functions," in *Advances in Neural Information Processing Systems II,* ed. D.S. Touretzky, Morgan-Kaufmann.

Obradovic, Z. and Parberry, I., (1989) "Analog neural networks of limited precision II: Learning with multilinear threshold functions (preliminary version)," *Technical Report CS-89-15,* Dept. of Computer Science, Penn. State Univ.

Parberry, I., (1990) "A Primer on the Complexity Theory of Neural Networks," in *Sourcebook of Formal Methods in Artificial Intelligence,* ed. R. Banerji, North-Holland, pp. 217-268.

OTHER TOPICS

The General Utility Problem in Machine Learning

Lawrence B. Holder
University of Illinois
Beckman Institute
405 North Mathews, Urbana, IL 61801

Abstract

Experiments have revealed that uncontrolled application of the analytical learning paradigm results in knowledge having low utility. Because the performance element must consider low utility knowledge along with high utility knowledge, the proliferation of low utility knowledge eventually defeats the goal of improved performance. Experiments in empirical learning have demonstrated a similar phenomenon. Uncontrolled application of an empirical learning paradigm may result in inaccurate knowledge, and a post-processing stage is typically needed to repair the degradation in performance. The results from experimentation in both analytical and empirical learning imply a general utility problem in machine learning. This paper presents evidence for such a perspective and recommends a closer dependence between the learning paradigm and the performance goals for which it is designed. A new approach is presented along with experimentation that illustrates the applicability of the approach to the general utility problem.

1 Introduction

One of the main goals for research in machine learning is the eventual integration of learning methods with knowledge-based systems. Learning methods offer the ability to transform the knowledge of the system to improve performance on the tasks using the knowledge. The ability to transform knowledge will reduce the dependency of system performance on the quality of the knowledge initially entered by the knowledge engineer.

Recent experimentation with machine learning methods has uncovered a new obstacle to their integration with knowledge-based systems. Experiments with several analytical learning systems demonstrated an eventual degradation in performance due to uncontrolled application of the learning paradigm. This obstacle has been named the *utility problem* [Minton88]. Experiments with empirical learning methods are also uncovering this phenomenon of performance degradation due to unconstrained application.

Two additional obstacles block the integration of current learning methods with knowledge-based systems. First, the performance goals for the tasks using the knowledge may change over time. Knowledge transformations made by machine learning methods must adapt to changes in the performance goals for the desired tasks. Second, the knowledge may support different tasks from multiple domains. Knowledge transformations to improve performance on one task must preserve the performance goals of other tasks using the knowledge. These additional constraints along with the original utility problem combine to form the general utility problem. The general utility problem in machine learning is the degradation of performance for tasks using the knowledge due to the unconstrained transformation of the knowledge by machine learning methods.

Recent solutions to the utility problem have generally followed the trend of applying the learning method to every task and then pruning away the knowledge that eventually turns out to degrade performance. This research offers a different approach called performance-driven knowledge transformation that selectively applies learning methods only when necessary to achieve a desired performance goal for some task. The approach acquires knowledge for controlling the application of multiple learning methods.

The next section reviews work related to the utility problem in analytical learning and casts recent work in empirical learning in the context of the utility problem. Section 3 defines the general utility problem for machine learning and discusses alternative solutions. Section 4 describes the performance-driven knowledge transformation approach to solving the general utility problem. Section 5 illustrates experiments performed with an implementation of the approach in the PEAK system. Section 6 concludes with plans for further improvements to the proposed approach.

2 Utility Problem in Learning

Research in both empirical and analytical learning has uncovered deficiencies in the employed methodologies. The major deficiencies stem from the naive view that the methodology in question is always applicable to the learning task and therefore should always be ap-

plied to the data. In a performance-driven system, one methodology is rarely sufficient to handle the variety of learning tasks.

2.1 Analytical Learning

Research on analytical (explanation-based) learning techniques began to focus more attention on performance with the appearance of Keller's work on the definition of operationality [Keller88]. Analytical techniques learn from a single example by proving the example is an instance of the concept to be learned. The proof terminates when the leaves of the proof tree are all operational predicates. The proof tree is then generalized, yielding an operational description of the concept. Earlier work on explanation-based learning defined an operational concept as one whose description is composed from a set of predicates deemed easy to evaluate [DeJong86, Mitchell86]. Keller pointed out that operationality is more intimately related to the performance element and the desired performance improvement. This view of operationality was used in the METALEX system that learns heuristics for solving calculus problems. METALEX defines an operational concept as one that improves the performance element's (problem solver's) run-time efficiency on a set of benchmark calculus problems, while maintaining effectiveness so that some percentage of the problems are still solved correctly. The increased attention on performance has led to the reevaluation of several analytical learning systems and the observation that performance may degrade with repeated application.

2.1.1 PRODIGY

In experimentation with the MORRIS analytical learning system, Minton found that performance degrades as the number of rules grows large [Minton85]. In order to learn a concept, the system acquires several rules whose disjunction forms the system's understanding of the concept. As the number of rules increase, the cost of determining the applicability of a rule may outweigh the benefits of applying, and thus, retaining the rule. Minton calls this phenomenon the *utility problem* and offers the PRODIGY system as a solution [Minton88]. PRODIGY maintains empirical estimates of match costs, application savings and frequency of application for each rule. These estimates are used to compute a utility value for the rule. If this value becomes negative, the rule is no longer considered. Minton found that maintenance of a rule's utility value and compression of the rule's conditions result in a substantial performance improvement. These results indicate that a system should be sensitive to the cost and savings of the learned descriptions.

2.1.2 SOAR

Experimentation on the SOAR system has uncovered similar results [Tambe88]. Instead of monitoring the cost and benefits of rules, Tambe and Rosenbloom restrict the expressiveness of the learned rules so that the complexity of the match is kept linear in the number of matching conditions [Tambe89]. Results of using this technique within SOAR indicate a greater number of less expressive rules are needed to attain the generality of the more expressive rules, but the match cost is no longer exponential. However, the results are unclear on whether an exponential number of simpler rules will be needed to achieve the generality of the more expressive rules. Also, the trend toward generating ground instances of the general rules seems contradictory to the purported benefits of analytical learning.

2.1.3 EGGS

Despite the aforementioned evidence for degrading performance in analytical learning systems, other such systems have demonstrated improved performance without concern for the number or form of the learned rules. Looking at systems by O'Rorke [O'Rorke87] and Shavlik [Shavlik88] Mooney recently uncovered the reason for the contradictory results [Mooney89]. The performance element for Mooney's experiments was EGGS [Mooney86], which includes a Horn-clause theorem prover and standard explanation-based learning techniques [DeJong86, Mitchell86] for generalizing the proofs.

Experiments with EGGS revealed that limited use of the learned rules provided greater performance in accuracy and speed than full use. Because Shavlik constrained the proofs to be no longer than a specified depth bound, his system was making only limited use of the learned rules (i.e., only those rules that required limited chaining).

Mooney also demonstrated that using a breadth-first search for theorem proving, instead of depth-first, also forced a limited use of learned rules. Learned rules that would have required deep sub-goaling to reach a solution are circumvented by the simultaneous consideration of proofs from the original domain theory. The use of breadth-first search in O'Rorke's system accounts for much of the favorable performance. Mooney concludes that limited use of learned rules is advisable until the system has learned the rules necessary to solve the more common problems.

2.1.4 Summary

Experimentation on analytical learning systems demonstrates performance degradation with uncontrolled application of the paradigm. In response, many researchers have opted for more specific instances of the learned rules. The level of specificity of the knowledge should not be arbitrary, but determined by the desired performance. In fact, with performance-directed learning the original domain theory may perform within desired performance thresholds, in which case, the application of analytical learning may be unnecessary.

2.2 Empirical Learning

Empirical learning methods have traditionally been designed to achieve the best classification performance possible. However, experimentation described below indicates that classification performance can actually degrade with repeated application of the empirical learning method.

2.2.1 AQ

During experimentation with the AQ system (specifically, AQ15 [Michalski86]), Michalski found that repetitive application of AQ can yield less accurate concepts than a more conservative application strategy combined with a simple inference mechanism [Michalski87]. The AQ methodology finds a conjunctive description that covers as many positive examples as possible without covering any negative examples. Positive examples not covered by the first description are used as input for another execution of AQ. This procedure continues until a concept in disjunctive normal form is produced covering all the positive examples and none of the negative examples. Michalski compared the accuracy of the DNF concept with that of the concept consisting of only the single disjunct covering the most positive examples. Using a simple matching procedure, the truncated concepts out-performed the original concepts in both accuracy and speed. This observation illustrates the need for systems to be more selective in their own behavior when such selectivity is sufficient to achieve the desired performance goals.

2.2.2 ID3

Similar results have been obtained with the decision trees generated by Quinlan's ID3 program [Quinlan86]. Quinlan found that *pruning* the rules extracted from a decision tree can improve the accuracy of the rules on unseen examples [Quinlan87]. ID3 builds decision trees by selecting an attribute from the training examples providing the best split (according to an information theoretic criterion) between positive and negative examples. The program continues by descending each branch and recursively applying itself to the examples satisfying the attribute value for that branch. ID3 halts when all the nodes at the frontier of the tree contain all positive or all negative examples. The pruning stage removes rules from the decision tree until accuracy on a set of test examples begins to decrease. Compared to the original rules, the pruned rules performed better on a set of unseen test examples. Although the success of pruning is due to noise, missing values, and the decreased number of examples available at higher depths in the tree, this stage might have been unnecessary if the desired accuracy had been taken into account during the initial generation of the decision rules.

2.2.3 Summary

Research on empirical learning has shown that inexact rules combined with a simple matching procedure can be less expensive and more accurate than exact rules. The tradeoff between accuracy and completeness of the learned rules should be decided by the desired performance. Modifying the AQ algorithm to return only the one disjunct covering the most positive examples may avoid generation of low utility knowledge. Likewise, constraining the ID3 algorithm to stop when the leaves have reached a desired level of accuracy may prevent inaccuracies at greater depths in the decision tree and avoid the need for pruning.

Typically, empirical learning methods are invoked to achieve the best classification accuracy possible. To avoid a degradation in classification performance, empirical learning methods should be invoked only when necessary to achieve a violated performance goal. Furthermore, repair of the performance goal violation may require only modest generalization as opposed to the large inductive leaps made by most empirical learning methods. In the extreme case (e.g., small number of instances in the concept), rote learning may be preferable to more powerful empirical learning methods.

3 General Utility Problem

The generation of low utility knowledge by both analytical and empirical learning methods indicates a utility problem in machine learning more widespread than that identified in the analytical learning literature. The general utility problem encompasses not only the performance degradation on one task due to uncontrolled application of learning methods, but also adaptation to changing task performance goals and maintenance of performance on other tasks using the knowledge. Thus, the general utility problem is informally defined as follows:

> **General Utility Problem:** performance degradation on one or more tasks due to the transformation of knowledge.

In order to address the general utility problem, the role of machine learning methods must be viewed from a purely performance-based perspective. This perspective is similar to that used by Keller in the METALEX system [Keller87]. METALEX transforms its knowledge base by retaining a new concept only if the concept improves the efficiency and effectiveness of the performance element.

Markovitch and Scott address the general utility problem by filtering the information flow from instances, to the knowledge base, and then to the performance element [Markovitch89]. One filter, the *utilization* filter, removes harmful rules from the knowledge used by the performance element.

Learning control knowledge for the application of learning methods to different tasks is addressed by Rendell's variable-bias management system (VBMS) [Rendell87]. VBMS maps different tasks to points in a bias space. Each point in bias space represents a choice of inductive algorithm, representation language

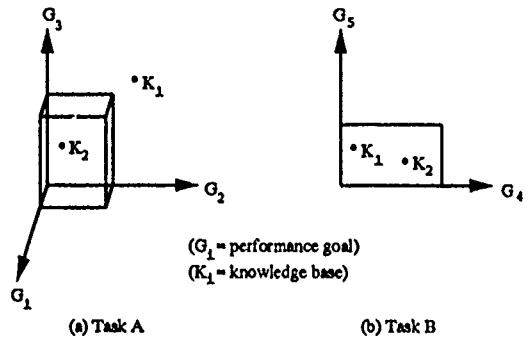


Figure 1: Performance Spaces for Two Tasks

and any relevant parameters for the algorithm or language.

The next section describes an approach to the general utility problem called performance-driven knowledge transformation. This approach differs from the approach in METALEX by making performance goals more explicit and incrementally adapting to changes in desired performance. In contrast to the knowledge filtering approach, performance-driven knowledge transformation constrains the initial generation of knowledge as directed by failure performance goals. This approach differs from the approach in VBMS in that the emphasis is on selecting learning methods to repair violations in desired performance, not to achieve maximum possible performance on an isolated learning task.

4 Performance-Driven Knowledge Transformation

Performance-driven knowledge transformation controls the application of learning methods based on their ability to achieve desired performance goals on one task while preserving the performance on other tasks. Each task for the knowledge base defines a performance space. The dimensions of the performance space are the performance goals (e.g., completeness, correctness, response time) to be maintained by the knowledge base for that task. The current state of the knowledge base is represented by a point in the performance space for each task. A knowledge transformation can be viewed as a move of the current knowledge base from one point in the performance space of each task to another. Figure 1 shows the performance spaces for two tasks. Task A (Figure 1a) consists of three performance goals G_1, G_2 and G_3. Task B (Figure 1b) consists of two performance goals G_4 and G_5. The location of two knowledge bases K_1 and K_2 are shown for each task.

The desired performance for each task defines a hyper-rectangle in that task's performance space. When the knowledge base moves outside the desired-performance hyper-rectangle in some performance space, performance-driven knowledge transformation selects a learning method to transform the knowledge base so that the corresponding point in the performance space for the current task moves back inside the desired-performance hyper-rectangle without moving the point outside the desired hyper-rectangle in the performance spaces for other tasks. Referring to Figure 1, knowledge base K_1 has satisfactory performance for task B, but violates the performance goals of task A. Transforming knowledge base K_1 to K_2 achieves the performance goals of task A and preserves the satisfactory performance for task B.

This research proposes an approach to performance-driven knowledge transformation implemented in the PEAK system. When a performance goal violation is detected while solving a problem from some task, the PEAK system uses information about the context of the goal violation (e.g., the difference between desired and actual performance) to select a transformation operator for reducing this difference while maintaining other performance levels. Application of the operator yields a new knowledge base. If the new knowledge base achieves the violated performance goal and preserves other performance goals, then the current knowledge base is replaced by the new knowledge base. Otherwise, another transformation operator is selected for application. Verification of the new knowledge base is accomplished by using the knowledge to solve previously seen problems from the same task. For each operator, PEAK retains information about the applicability of the operator in a given context based on the success of the operator in reducing the goal violation. As more performance goal violations are repaired, PEAK demonstrates more intelligent selection of transformation operators and quicker convergence to a knowledge base within desired performance thresholds.

In the following discussion, certain assumptions have been made about the knowledge in the knowledge base and the performance element using this knowledge. The knowledge base is a set of Horn clause rules. The performance element is a deductive retriever similar to Prolog. Performance is measured while the performance element attempts to solve a query posed by the user. Attached to the query are the performance goals to be maintained during the solution of the query. Performance goal violations occur when the measured performance exceeds the desired thresholds.

4.1 Performance Perspective

Using performance goals as a means of guiding the maintenance and repair of a knowledge base requires a precise definition of performance. The definition of performance depends on the perspective. Four perspectives are applicable for describing the performance of a knowledge base:

- **External** performance is the performance measured from outside the knowledge base, regardless of any internal knowledge transformations.

- **Current** performance is the performance the system currently maintains for the previously seen queries.
- **Expected** performance is the performance the system expects to demonstrate on future queries. Expected performance is usually the same as current performance.
- **Absolute** performance is the performance that the current state of the knowledge would support if given every possible query.

When the user specifies a threshold for some performance measure, the proper perspective must be used to evaluate the performance of the knowledge base. *Absolute* performance is rarely available due to a lack of knowledge about the instance space. *Absolute* performance is inappropriate, because the distribution over the entire instance space may not give equal probability to each instance. *External* performance provides information about the rate of convergence towards absolute performance. Changes in *external* performance indicate the need for an increase or decrease in the extent of the knowledge transformations. *Current* performance evaluates the knowledge only on previously seen queries. *Expected* performance is the best measure of the current state of the knowledge base, because the objective of the knowledge base is to maintain its expected ability to perform the task within desired thresholds on possibly unseen queries.

Performance-driven
knowledge transformation should measure both Both *expected* and *external* performance should be measured by the performance-driven knowledge transformation process. Knowledge transformations are triggered only when *expected* performance falls below desired levels. *External* performance should then be used in the selection of an appropriate transformation operator. The greater the difference between *external* and *expected* performance, the more drastic a transformation operator should be recommended by the system.

4.2 Information on Goal Violations

Once a goal violation has been detected, several pieces of information are available for selecting an appropriate knowledge transformation operator. First, as described in the previous section, the difference between expected and external performance indicates the extent of the necessary transformation.

Second, after the performance element attempts to solve a query, the violated and preserved goals are known. Each goal contains information about the performance measure that this goal constrains, the desired threshold on the measure, the observed value of the measure on previously seen queries (including the query just processed), and the difference between the observed and desired performance (the error). The performance measure constrained by a violated goal is useful for selecting transformation operators capable of improving this performance measure. The magnitude of the error indicates the extent of the transformation. The performance measure constrained by a satisfied goal is useful for selecting transformation operators capable of preserving this performance measure. The magnitude of the error indicates the extent to which the selected operator may degrade performance on the satisfied goals in order to achieve performance on the violated goals.

A third source of information that will be available upon detection of a performance goal violation is the *task history*. Each task known to the knowledge base maintains a task history of previously seen queries from the task. The task history serves two purposes. First, the task history represents an empirical estimate of the distribution over the possible queries of the task. This distribution can be used to verify the achievement of violated performance goals in transformed knowledge. Second, an entry in the task history contains information about the query-solving episode. One useful piece of information about a query-solving episode is the trace of the knowledge accessed during the solution.

The *knowledge trace* is an and/or tree that records the knowledge accessed during the solution of the query and indicates which rules (if any) support the response to the query. Information about the shape of a task's knowledge traces constrains the selection of knowledge transformations. For example, wide, shallow knowledge traces indicate that the knowledge consists of specific instances of the task; whereas narrow, deep knowledge traces indicate a more general set of rules for proving queries from the corresponding task

Finally, past success of the transformation operators provides information upon performance goal violation. As the knowledge base transforms to meet performance goals, a record is kept of the old and new knowledge bases along with the operator responsible for the transformation. If the new knowledge base achieves a violated goal while preserving non-violated goals, then the system increases the operators applicability for achieving and preserving the appropriate goals. Over time, collection of this inform 'ion will allow the system to make a more informed operator selection based on past experience.

4.3 Verification of Knowledge Base

Because no operator application is guaranteed to achieve the desired results, the system must verify that the knowledge base resulting from an operator application achieves the desired performance. Verification can be accomplished by re solving the queries in the task history. The size of the task history can be changed to tradeoff performance convergence rates for transformation speed. As the system learns operator applicability, there is less chance that several alternative operator applications must be tried before finding one that achieves the violated goal. Because each transformation attempt requires verification of the resulting knowledge base, the fewer attempts necessary

implies fewer verifications; thus, the task history size can increased over time.

5 Experimentation

This section illustrates the application of PEAK on two tasks from diverse domains. The first experiment involves learning to improve response time, completeness and correctness while determining whether to land the space shuttle manually or automatically depending on environmental conditions. The second experiment involves learning to improve response time while constructing plans to build towers in the blocks-world domain. Together, the two experiments demonstrate the ability of performance-drive knowledge transformation to selectively apply appropriate learning methods to achieve desired performance goals.

5.1 Shuttle Domain

This experiment executes the PEAK system on the shuttle landing control database available from the machine learning databases maintained by University of California at Irvine. The problem is to determine whether to land the shuttle manually or automatically based on environmental attributes. The corresponding task is labeled the *landing* task, and the queries are of the form `landing(ENV,?x)`. The `ENV` in the query represents the environmental situation to be evaluated. The performance element attempts to fill in the `?x` with the recommended landing control: `auto` or `noauto`.

Prior to query answering, the user inputs the performance thresholds to be maintained by the knowledge base while answering *landing* queries using the performance element (a backward-chaining deductive theorem prover for Horn clauses). For this experiment, three performance goals are specified: *correctness*, *completeness* and *response time*. The correctness goal specifies that the answers to queries must be correct 90% of the time. The completeness goal specifies that the query must be answered 95% of the time. That is, the answer should be either auto or noauto and not *"I don't know"*. The response time goal specifies that the performance element must respond within 10 seconds.

Two knowledge transformation operators are available: rote learning and empirical learning. Application of the rote learning operator asks the user for the correct answer to the query. A new rule is added to the knowledge base having the instantiated query as the consequent, and the facts defined before query execution as the antecedent. The empirical learning operator utilizes the ID3 program to build a decision tree from examples in the knowledge base. The examples are rules such as those learned by the rote operator. Each path in the resulting decision tree is converted to a rule. The examples are replaced by the new rules in the transformed knowledge base.

Starting with an empty knowledge base, PEAK attempts to solve *landing* queries, while maintaining the performance goals. Figure 2 plots the three performance goals for 200 randomly chosen queries from the shuttle landing control domain.

Figure 2a illustrates how PEAK maintains response time performance below 10 seconds. For the first 30 queries, response time increases as the number of rote-learned rules increases. Eventually, the large number of rules in the knowledge base cannot be traversed within the response time threshold.

While processing the 30th query, PEAK was unable to solve the query, generating a completeness failure. PEAK first trys to transform the knowledge base by rote-learning a new rule. However, verification of the new knowledge base uncovers a response time failure. Because the rote learning operator was ineffective, PEAK chose to apply the ID3 operator. ID3 generalized the 29 learned instances into 8 general rules. As Figure 2a indicates, the resulting transformation drastically improves response time performance.

The plot of completeness performance in Figure 2b illustrates how PEAK quickly learns the initial query knowledge. After the ID3 transformation, completeness remained above the 95% threshold for the remainder of the 200 queries.

The correctness plot in Figure 2c shows how performance starts at 100% and converges to the desired 90% threshold. The initial values of 100% for correctness are due to the fact that many of the initial queries could not be answered. Correctness performance only measures the correctness of answered queries. Immediately following the application of ID3, correctness falls to 94% due to the next two queries being incorrectly answered according to the new knowledge base. As query answering continues, the over-generalization in the rules eventually brings correctness down below the 90% threshold. Correctness violations occur at queries 89, 98, 153 and 163. In each case, PEAK uses the rote-learning operator to memorize the incorrectly answered query and restore 90% correctness performance.

The final knowledge base after completion of the 200 queries consists of the 12 rules shown in Figure 3. Rules 5-12 are the general rules learned by ID3. Rules 1-4 are the specific instances learned to repair the over-generalization in ID3's rules. After 200 queries, the knowledge base converged to 8 general rules describing major trends in the shuttle landing domain and four specific rules for special cases not handled correctly by the general rules.

One final observation from Figure 2 is the convergence of the performance towards the desired thresholds and not towards the maximum possible performance. This indicates how performance-driven knowledge transformation utilizes flexibility in one dimension of performance to improve performance in another dimension.

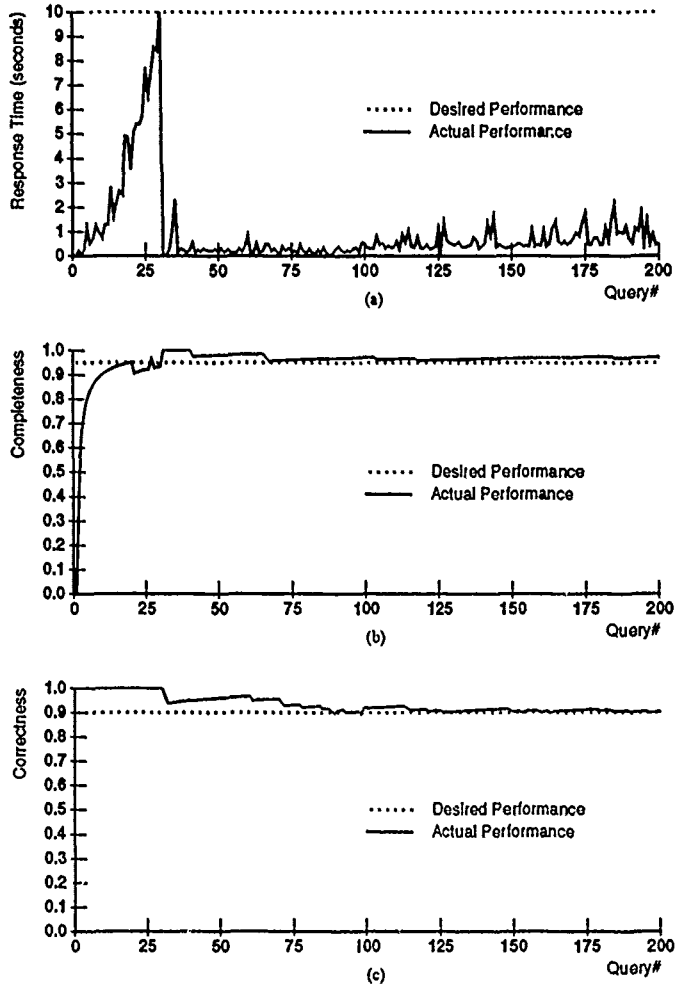


Figure 2: Plots of Performance for Shuttle Domain

1. landing(x,noauto) ← sign(x,nn) & wind(x,head) & stability(x,xstab) & error(x,MM) & magnitude(x,Medium) & visibility(x,yes)
2. landing(x,noauto) ← sign(x,pp) & wind(x,tail) & stability(x,xstab) & error(x,MM) & magnitude(x,Low) & visibility(x,yes)
3. landing(x,noauto) ← sign(x,nn) & wind(x,head) & stability(x,stab) & error(x,MM) & magnitude(x,OutOfRange) & visibility(x,yes)
4. landing(x,noauto) ← sign(x,nn) & wind(x,tail) & stability(x,xstab) & error(x,MM) & magnitude(x,Low) & visibility(x,yes)
5. landing(x,auto) ← error(x,MM) & visibility(x,yes)
6. landing(x,auto) ← stability(x,stab) & error(x,SS) & magnitude(x,Strong) & visibility(x,yes)
7. landing(x,auto) ← visibility(x,no)
8. landing(x,noauto) ← error(x,XL) & visibility(x,yes)
9. landing(x,noauto) ← error(x,LX) & visibility(x,yes)
10. landing(x,noauto) ← stability(x,xstab) & error(x,SS) & magnitude(x,Strong) & visibility(x,yes)
11. landing(x,noauto) ← error(x,SS) & magnitude(x,OutOfRange) & visibility(x,yes)
12. landing(x,noauto) ← error(x,SS) & magnitude(x,Low) & visibility(x,yes)

Figure 3: Shuttle Domain Knowledge Base After 200 Queries

5.2 Blocks-World Domain

In the task from the blocks-world domain, the user asks the performance element to construct a plan for building a tower of blocks. The queries are of the form `tower(A B C ?state)`, where `A`, `B` and `C` are blocks, and `?state` is a variable to be instantiated with the plan for achieving the tower.

Prior to query answering, the user inputs the performance thresholds to be maintained by the knowledge base while answering *tower* queries. For this experiment, one performance goal is specified: response-time < 10 seconds. Performance goals for completeness and correctness are inappropriate, because the domain theory is assumed complete and correct.

In addition to the rote learning and ID3 operators used with the first experiment, an explanation-based generalizer, EGGS [Mooney86], is included in the PEAK system. EGGS applies standard explanation-based techniques [DeJong86, Mitchell86] to generalize the proofs obtained by the performance element. When EGGS is applied to a proof, the result is a general rule that is added to the knowledge base.

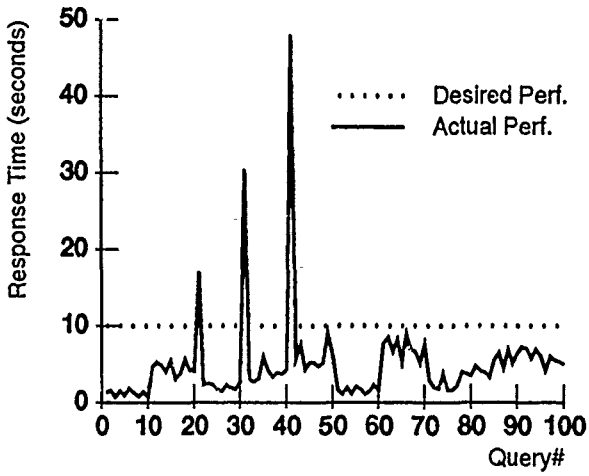


Figure 4: Plot of Response Time for Tower Domain

Starting with the blocks-world domain theory, PEAK attempts to solve *tower* queries, while maintaining the response time performance goal. The initial state of the blocks world contained six blocks. Figure 4 shows the response time obtained by PEAK for 100 semi-randomly chosen *tower* queries. Semi-random means that the first ten queries were all towers of height two (the two blocks to be in the tower were chosen randomly from the six blocks in the initial state). The second ten queries were all towers of height three, and so on for the first 50 queries. The second 50 queries repeat the above sequence to show the effects on response time of the rules learned during the first 50 queries.

As shown in Figure 4, the first 20 queries (towers of height two and three) are solved by the original domain theory within the response time threshold. However, the domain theory is unable to maintain the response time performance goal while processing the 21st query (tower of height three). At this point, ID3 cannot be applied due to the lack of examples in the knowledge base. The EGGS operator is chosen over rote learning due to the knowledge trace for the query. The deep, wide proof tree suggests that EGGS is more likely to succeed than ID3.

Application of EGGS yields a general rule that builds any tower of height three in one step. Thus, the remainder of the *tower* queries for height three are completed within the response time threshold. Similar rules are learned for the 31st query (tower of height four) and 41st query (tower of height five). Figure 4 shows that retrying the towers of heights two through five (queries 50-100) results in no response time performance violations due to the previously learned rules.

This experiment demonstrates PEAK's ability to constrain the application of the EGGS analytical learning algorithm. Application of EGGS was unnecessary for towers of height two and three, because the original domain theory was able to solve these queries within the desired performance thresholds. However, the domain theory was unable to support the desired performance for towers of height four, five and six, requiring three applications of EGGS to learn general rules for these specific cases. As the performance on the second 50 queries indicates, the original domain theory plus the three learned rules was sufficient to maintain the desired performance for the tower-building task.

6 Conclusions

In order to integrate machine learning methods with knowledge-based systems, the general utility problem in machine learning must be addressed. Evidence for the general utility problem has been found in experimentation on both analytical and empirical learning methods. Unconstrained application of these methods has been shown to degrade the performance they were designed to improve. Learning methods should be invoked only after a performance failure has been detected, and then, only if the learning method is applicable to the properties of the failure. Furthermore, learning methods must permit transformation of the knowledge to achieve performance goals without violating the performance goals associated with other tasks using the knowledge. The learning methods must also have the ability to adapt to changing performance goals.

Performance-driven knowledge transformation offers an approach that addresses the general utility problem. Learning methods are invoked only when necessary to improve performance, and in accordance with previous success in repairing the violated performance goal. The performance-driven knowledge transformation approach has been implemented in the PEAK sys-

tem. Experimentation with PEAK demonstrates the ability to control the application of learning methods to achieve desired performance goals.

More experimentation is necessary to validate the use of performance-driven knowledge transformation. Experiments with modified versions of the AQ and ID3 algorithms will indicate the usefulness of these transformation methods to avoid inaccuracies due to unconstrained application of the paradigms. Experiments with the interaction of multiple learning methods will indicate the usefulness of current control knowledge and suggest the need for other knowledge to control the performance-driven knowledge transformation process.

7 Acknowledgments

I would like to thank Robert Stepp for his insightful comments on this work. I would also like to thank Diane Cook, Brad Whitehall and Robert Reinke for their helpful suggestions. The ID3 and EGGS operators were adapted from versions written by Ray Mooney. The blocks-world domain theory was adapted from rules written by Jude Shavlik and Ray Mooney.

8 References

[DeJong86] G. F. DeJong and R. J. Mooney, "Explanation-Based Learning: An Alternative View," *Machine Learning 1*, 2 (April 1986), pp. 145-176.

[Keller87] R. M. Keller, "Concept Learning in Context," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 91-102.

[Keller88] R. M. Keller, "Defining Operationality for Explanation-Based Learning," *Artificial Intelligence 35*, 2 (June 1988), pp. 227-241.

[Markovitch89] S. Markovitch and P. D. Scott, "Utilization Filtering: A Method for Reducing the Inherent Harmfulness of Deductively Learned Knowledge," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 738-743.

[Michalski86] R. S. Michalski, I. Mozetic, J. Hong and N. Lavrac, "The Multi-Purpose Incremental Learning System AQ15 and its Testing Application in Three Medical Domains," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 1041-1047.

[Michalski87] R. S. Michalski, "How to Learn Imprecise Concepts: A Method for Employing a Two-Tiered Knowledge Representation in Learning," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 50-58.

[Minton85] S. Minton, "Selectively Generalizing Plans for Problem-Solving," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 596-599.

[Minton88] S. Minton, "Quantitative Results Concerning the Utility of Explanation-Based Learning," *Proceedings of the National Conference on Artificial Intelligence*, St. Paul, MN, August 1988, pp. 564-569.

[Mitchell86] T. M. Mitchell, R. Keller and S. Kedar-Cabelli, "Explanation-Based Generalization: A Unifying View," *Machine Learning 1*, 1 (January 1986), pp. 47-80.

[Mooney86] R. J. Mooney and S. W. Bennett, "A Domain Independent Explanation-Based Generalizer," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 551-555.

[Mooney89] R. J. Mooney, "The Effect of Rule Use on the Utility of Explanation-Based Learning," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 725-730.

[O'Rorke87] P. V. O'Rorke, "LT Revisited: Experimental Results of Applying Explanation-Based Learning to the Logic of Principia Mathematica," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 148-159.

[Quinlan86] J. R. Quinlan, "Induction of Decision Trees," *Machine Learning 1*, 1 (1986), pp. 81-106.

[Quinlan87] J. R. Quinlan, "Generating Production Rules from Decision Trees," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 304-307.

[Rendell87] L. Rendell, R. Seshu and D. Tcheng, "Layered Concept Learning and Dynamically-Variable Bias Management," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 308-314.

[Shavlik88] J. W. Shavlik, "Generalizing the Structure of Explanations in Explanation-Based Learning," Ph.D. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, January 1988.

[Tambe88] M. Tambe and A. Newell, "Some Chunks Are Expensive," *Proceedings of the 1988 International Machine Learning Workshop*, Ann Arbor, MI, June 1988, pp. 451-458.

[Tambe89] M. Tambe and P. Rosenbloom, "Eliminating Expensive Chunks by Restricting Expressiveness," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 731-737.

A Robust Approach to Numeric Discovery

Bernd Nordhausen and Pat Langley*
Department of Information & Computer Science
University of California, Irvine, CA 92717 USA

Abstract

IDS is an integrated discovery system that forms taxonomic hierarchies, notes qualitative relations, and finds numeric laws. This paper focuses on the system's algorithm for numeric discovery. We describe the basic method in terms of heuristic search through a space of numeric terms, provide a simple example, and show how IDS uses the algorithm to find three classes of numeric relations. We then evaluate the system's law-finding ability through experiments with artificial domains, examining the effect of system parameters, noise level, number of irrelevant terms, and law complexity on both asymptotic accuracy and learning rate. Finally, we consider the algorithm's relation to other numeric methods and outline directions for future work.

1 Introduction

The discovery of numeric laws is a central part of the scientific process. In this paper we focus on the numeric discovery component of IDS (Nordhausen, 1989), an empirical discovery system. In the following sections, we detail the system's algorithm for finding quantitative relations, and then report experiments with artificial domains to evaluate the algorithm's ability to find laws under a variety of conditions. Finally, we consider some directions for future research and discuss the method's relation to earlier work. However, let us first briefly describe the overall system in which this algorithm plays a role.

IDS is an integrated discovery system that represents observations and laws using Forbus' (1985) qualitative process formalism. Each observed *history* consists of a temporal sequence of qualitative states, which represent intervals during which the signs of derivatives remain constant and during which the structure remains unchanged. The system employs an incremental clustering method, similar to Lebowitz' (1987) UNIMEM and Fisher's (1987) COBWEB, to classify new qualitative states and incorporate them into a taxonomic hierarchy. IDS also stores links that indicate temporal succession among states, along with transition conditions between pairs of states. Moreover, the system forms generalized links that connect abstract qualitative states in its hierarchy; together with the transition conditions, these constitute an important class of qualitative laws.

2 Numeric Discovery in IDS

In addition to the above activities, IDS searches for numeric laws to augment its qualitative descriptions. These may specify the conditions for moving from one state to another, a relation between numeric attributes within a given qualitative state, or a quantitative relation between variables in different states. Each of these cases involves storing a law at a node or link in the taxonomy that summarizes information in the children of that node or link. Nordhausen (1989) describes the overall system in detail; in this paper we will limit our attention to numeric discovery.

2.1 The Basic Numeric Discovery Algorithm

IDS uses a single procedure to find all forms of numeric laws. Briefly, whenever the system adds a new qualitative state S (or transition link T) to an existing node (or link) in the hierarchy, it checks to see if S (or T) obeys the laws currently stored at the node. If not, IDS searches for new laws that cover the new child and its siblings, using the old law as a starting point.

For a given data set, the system attempts to find a law that covers these data by conducting a beam search through the space of numeric terms. More precisely, the search task can be stated as:

- *Given:* a set of base terms $a, b, c, \ldots$, along with one designated term (a) from that set;
- *Find:* a term $x = a^{n_0} \cdot b^{n_1} \cdot c^{n_2} \ldots$ such that a linear relation of the form $a = mx + n$ holds.

IDS searches from simple terms to more complex ones, using *correlation analysis* (Freund & Walpole, 1980) to direct the search process. As in Langley, Bradshaw, and Simon's (1983) BACON, the basic operators

*Current address: AI Research Branch, Mail Stop 244-17, NASA Ames Research Center, Moffett Field, CA 94035

Table 1. The IDS algorithm for finding numeric laws.

```
Variables: S is the set of base terms;
           D is the designated term;
           A is a defined term;
           C is the set of current terms;
           P and Q are sets of terms.

Find-numeric-law(D, S, C)
  Let A be the term in C that has the highest
      correlation with D.
  If the correlation between D and A is high enough,
     Then call linear regression on D and A
          to find the slope and intercept.
        Return A, the slope, and the intercept.
  Else if the maximum search depth is reached,
          then return the empty set.
  Else let C' be Find-best-terms(D, S, C).
       Find-numeric-law(D, S, C').

Find-best-terms(D, S, C)
  Let P be the products of the terms of S and C.
  Let Q be the quotients of the terms of S and C.
  For each term A in the union of P and Q,
      Compute the correlation between D and A.
  Return the terms with the N highest correlations.

Parameters:
  Width of the beam N (memory size);
  Threshold of the correlation (accuracy);
  Maximum degree of terms (law complexity);
  Maximum search depth (when to halt).
```

involve defining new terms as products and ratios of existing terms. The system initially examines correlations between the designated term and observable attributes, uses these to select promising products and ratios, and then recurses if it cannot find a law with the existing terms.

This search technique has a semi-incremental flavor. In cases where IDS has rejected an existing law, there is no need to reconsider the term in that law and those leading to the law. Thus, it uses the old term as the starting point for the new search, saving considerable effort over an approach that starts from scratch. However, this method does require that one store and reprocess all the data that led to the rejected law. As a result, it does not quite fit with the strict definition of an incremental learning system, though we hope to modify this in future versions of IDS.

Table 1 gives the basic algorithm for finding numeric relations. The top-level function, `find-numeric-law`, is given three arguments: the designated[1] term D, the set of base terms S, and a set of current terms C. If

[1] As detailed in Section 2.3, the system iterates through multiple base terms, treating each in turn as the designated term and searching for a law that predicts each one.

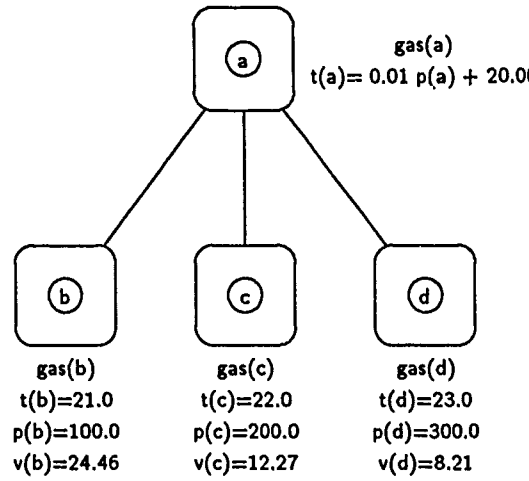


Figure 1. A spurious relation found during the rediscovery of the ideal gas law.

IDS is attempting to revise an existing law, C contains only the term occurring in the right-hand side of that law. If the system is searching for a new law, C is the set of observable terms S.

At each point in the search, IDS defines all of the products and ratios between the terms in the set S and those in C, but it retains only those terms having the highest correlations with the designated term D. These new terms become the current set C, and the function `find-numeric-law` is called recursively, with the designated term D and the base terms S remaining the same. If any term in C has a sufficiently high correlation with D, IDS ends the search and uses a regression technique to find the slope and intercept of the line relating them. The system continues in this fashion until it finds such a linear relation or until it exceeds the maximum search depth. If the search fails, IDS assumes that no law covers all the observed data. As we discuss in Section 3, the system includes four parameters that constrain its search for numeric laws.

2.2 An Example: Finding the Ideal Gas Law

As an example, let us consider how IDS rediscovers the ideal gas law. The system receives data in the form of states with gaseous objects at different temperatures, pressures, and volumes. Figure 1 shows the hierarchy after the system has processed three states, with all instances stored under a common parent node. Given these data, IDS finds a law relating the temperature and the pressure, because one can express the temperature as a linear function of the pressure. Now the system observes a fourth instance, which it adds as a child of Node 1 because it matches the parent completely.[2] However, this new instance violates the numeric law stored at the parent node, causing IDS to search for a new relation that covers all four instances.

[2] The system does not consider whether instances satisfy numeric laws during the clustering process.

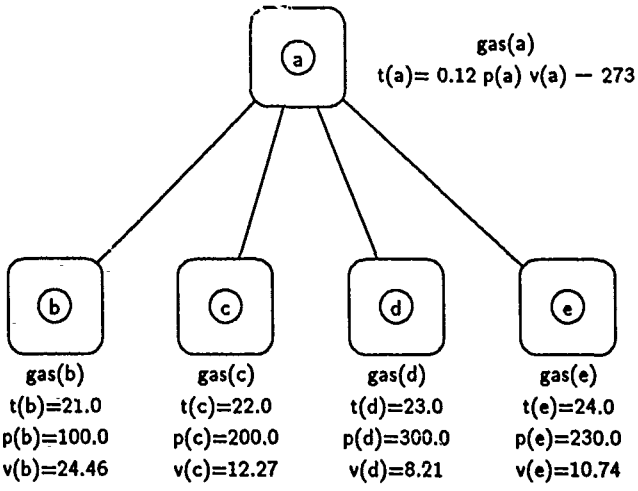


Figure 2. A correct version of the ideal gas law.

Because the term P was used in the rejected law, IDS calls the function `find-numeric-law` with {P} as the current set C, T as the designated term, and {P, V, T} as the set of base terms, S. In other words, IDS uses the term P as the entry point in the search space, starting by combining P with the terms in S to form products and ratios such as PV, P^2, P/T, and P/V. Of these new terms, PV has a high enough correlation to end the search. Regression produces the numeric law $T = 0.12 \times PV - 273$; this version is equivalent to the standard form of the law, $PV = 8.32(T + 273)$. Figure 2 shows the hierarchy that emerges after this revision is complete. As the system processes more instances and stores them under the parent node, it finds that they obey this new relation, so `find-numeric-law` is not called again.

IDS also defines new terms by a second method similar to that used in BACON. Whenever the system finds linear relations, it introduces the slopes and the intercepts of these relations as new numeric terms. Then IDS uses these new quantities to find more complex quantitative laws at higher levels in the state hierarchy. Consider again the discovery of the ideal gas law. In addition to the temperature (T), the pressure (P), and the volume (V), suppose IDS is also given the number of moles (N) of each gas. When the system observes gases with 1.0 mole at different temperatures, pressures, and volumes, it finds a linear relation between T and PV in the manner described above. At this point, IDS defines the slope s and the intercept i of the relation, which have values of 0.12 and −273, respectively.

Upon processing new states with different numbers of moles, the system calls on `find-numeric-law` with {PV} as the set of current terms. In this manner, IDS finds that T is linearly related to PV without search. Moreover, the system treats the slope and intercept (s and i) as higher-level dependent terms, calling on its numeric discovery method to find a law relating them to N. As a result, IDS finds that the intercept i is constant regardless of the number of moles N, but that the slope s is proportional to N. The system stores both laws at a more abstract state in the hierarchy, giving a set of relations equivalent to the standard version of the ideal gas law: $PV = 8.32\ N\ (T + 273)$.

2.3 Three Uses of the Basic Algorithm

Now that we have explained the basic method for discovering numeric laws, let us describe how IDS uses this algorithm to discover three different forms of such laws. The system augments the hierarchy of qualitative states with numeric relations in three ways, each of which serves a different purpose:

- A law within a state relates quantities that are constant within that state.[3]
- A numeric relation on a transition link specifies the numeric conditions for moving from the current state to its immediate successor.
- A numeric law between two states within the same qualitative history relates a quantity in a successor state (sometimes many steps ahead) to quantities in the current state.

Recall that `find-numeric-law` takes as arguments the designated term D, the set of base terms S, and the set of current terms C. The initial settings of these arguments differ in the three forms because they emphasize different quantities.

Thus, when IDS encounters the first case – finding a relation involving some quantity that occurs within a state – it calls `find-numeric-law` with that quantity as the designated term D, and the union of all within-state quantities and the transition quantities as the set of base terms S. The system repeats this process for each quantity in the state. Nordhausen (1989) reports a number of within-state relations that IDS finds in this manner, including the ideal gas law, the equality of final temperatures in heat-mixture experiments, and the constant ratios of molar concentrations that occur in reactions involving chemical equilibrium.

Similarly, when IDS attempts to find relations on a transition link between two states, a transition quantity becomes the designated term D, whereas the set of base terms is again the union of all within-state quantities and transition quantities. This lets the system discover the conditions for moving between qualitative states. Nordhausen (1989) recounts a variety of such discoveries, involving melting and boiling points, fluid flow phenomena, and the rates of chemical reactions. For instance, we presented IDS with a set of three-state histories in which two containers a and b begin with different levels of fluid L_a and L_b. After opening a conduit between the containers, one level increases

[3] When IDS is given a qualitative history, it is told which quantities are constant within each state and which are changing. The current system attempts to find numeric laws only for the former terms, but in principle the algorithm in Table 1 could also be used to discover relations between changing variables.

and the other decreases for a time, until both quantities stop changing simultaneously. Given these data, the system discovers that the transition from the second state to the third state occurs when $L_a = L_b$, that is, when the system reaches equilibrium. Transition laws can also involve simple constants, as in the case of melting and boiling points. Such relations let IDS implicitly specify inequalities, since they state that certain changes continue to occur until the transition conditions are met.

The system uses a forward propagation method in order to find numeric laws between qualitative states. It first attempts to relate a quantity in the current state to the quantities in the immediate successor state. If IDS cannot infer a law between a state and its immediate successor, it looks for a numeric relation between the state and the successor of the successor, continuing this chain until it finds a relation or it reaches a state with no successor.[4] Nordhausen (1989) presents numerous examples of across-state relations that IDS discovers, including Black's law of specific heat, the chemical law of combining weights, and Archimedes' principle of displacement. For instance, in the fluid-flow domain described above, the system finds a numeric relation $L_f = \frac{1}{2}L_a + I$, where L_a is the initial level of container a, L_f is the final level, and I is constant for a given value of L_b, the initial level of container b. Moreover, IDS finds a second law that holds at a more abstract level of its state and link hierarchy: $I = \frac{1}{2}L_b$. Taken together, these expressions can be rewritten as $L_f = \frac{1}{2}L_a + \frac{1}{2}L_b$, giving a more general law relating the three quantities.

3 Experimental Studies of Numeric Discovery

In this section, we evaluate the robustness of IDS' numeric discovery component using artificial domains. We first examine the effect of varying certain system parameters and then investigate how different levels of noise in numeric data influence the system's predictive accuracy. Finally, we study the effects of irrelevant attributes and law complexity on the learning rate.

3.1 Influence of the Correlation Threshold

IDS' numeric discovery component incorporates four system parameters. The *beam width* determines the memory size used in the search process, whereas the *maximum degree* of terms and the *maximum depth* of the search tree limit the amount of search. Experience has shown that these parameters do not have a substantial influence on the system, provided they are sufficiently large; for the experiments reported below, we used five as the value for the beam size and the maximum degree, and 12 as the value for the maximum depth. However, the fourth parameter, the *correlation threshold*, does impact the behavior of the system. IDS uses this threshold value as a termination criterion to end its search for numeric terms. For example, consider the law $x = a^2$. If the data contain no noise, then the correlation between x and a^2 is one. However, if there is noise in either x or a, then the correlation between x and a^2 will never be perfect. Hence, if the numeric data are likely to contain noise, then the correlation threshold must be set to a lower value.

In order to determine an optimal value for the correlation threshold in the presence of noisy data, we performed an experiment in which we varied this parameter. We randomly generated values for six numeric attributes (a, b, c, d, e, and x) obeying the relation $x = 2.0 \times \frac{abc}{d^2} + 30.0$. The range of variables a through e was 10.0 to 100.0, and values of x ranged from 30.2 to 20030.0. Attribute e was not used in the law, and thus was an irrelevant attribute. Furthermore, we introduced noise into the values of x, using constant inaccuracy with $\sigma = 100.0$, as described in Section 3.2. We gave IDS these data with the goal of finding a law that related x to the other numeric variables.

Because IDS is incremental, it generates a hypothesis for the law as it incorporates each instance. We then used this hypothesis to measure the accuracy of prediction using an independent noise-free test set. We chose noise-free rather than noisy test data to measure the accuracy of prediction because the former allows a standard control to compare the different experimental conditions. In this experiment, we measured the absolute difference between the predicted and actual values of x after every three instances (for efficiency reasons) and recorded the average difference over 30 test instances. IDS did not incorporate the instances of the test set, but only used them to measure the accuracy of the current hypothesis.

Figure 3a displays the learning curves for three different values of the correlation threshold (0.99, 0.98, and 0.97) averaged over 30 different runs. When the threshold was 0.99, IDS needed at most 18 instances to find the correct term ($\frac{abc}{d^2}$). It then continued to adjust the slope and intercept of the linear relation, which slowly approached the correct values and provided increasing predictive accuracy. When the correlation was 0.98, the system found the correct term in all but one run after at most 30 instances.[5] Using a correlation threshold of 0.97, the system found the correct term in 26 out of 30 runs after it had processed 75 instances. As IDS observes more instances, it will eventually find the correct term and predict the value with greater accuracy. Thus, it appears that the higher correlation thresholds yield better learning rates. However, we also ran the same data with 0.999 as the threshold value. In many runs, IDS failed to find a term that has a correlation higher than 0.999 within 75 training instances, for reasons that we will explain in Section 3.2.

[4] This search process would be expensive in complex domains, but the worst-case cost should increase only as the square of the length of the histories.

[5] The one incorrect term is the reason for the difference (after 30 instances) between the learning curves when the threshold is 0.99 and when the threshold is 0.98.

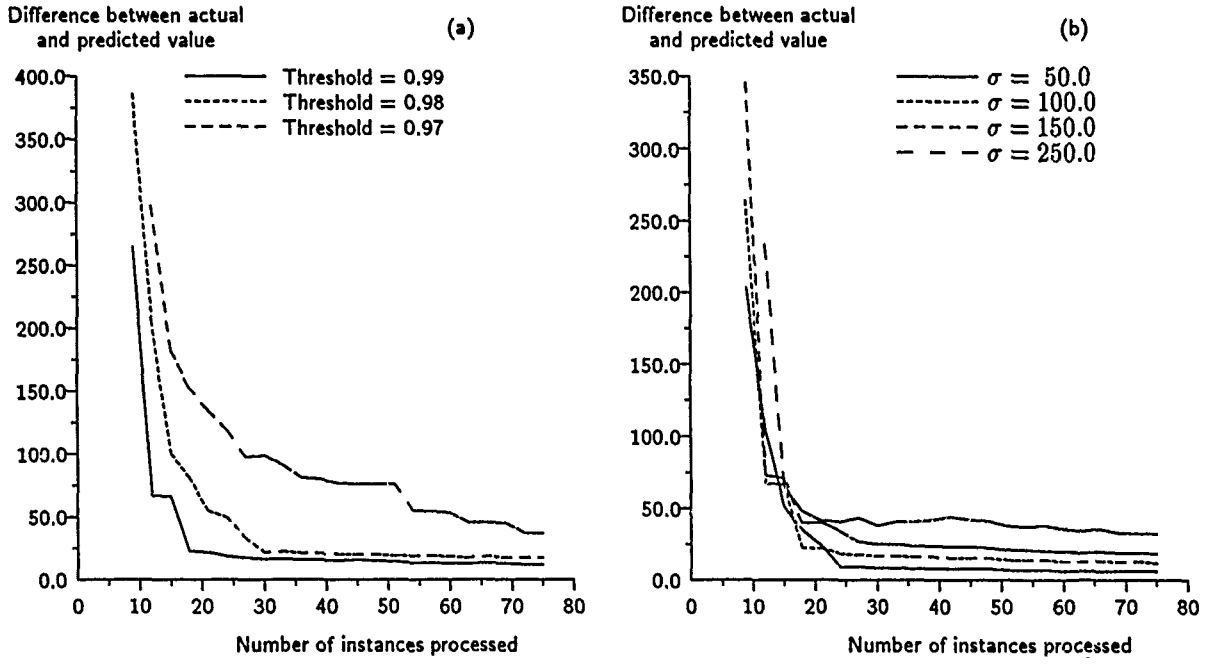


Figure 3. (a) The effect of varying IDS' correlation threshold on predictive ability. (b) The influence of noise due to constant inaccuracy on predictive ability.

Hence, the value of the correlation threshold impacts the behavior of the system. If the value is set too high, IDS will fail to find a term with an acceptable level of correlation. If the level is too low, the learning rate decreases. However, we will show in the next section that one correlation threshold value can be used with data over a wide range of noise levels. We will use a value of 0.99 as the correlation threshold for the remainder of this paper.

3.2 The Influence of Noise

Much of the early work on empirical discovery assumed that data were noise free (e.g., Lenat, 1978; Langley et al., 1983). This is clearly a simplification, and in this section we study how different levels of noise influence the behavior of IDS' numeric discovery algorithm. All numeric data in science ultimately come from measurement instruments, and one primary source of error stems from the inaccuracy of the instruments or the operator handling the instrument.

We considered two kinds of measurement inaccuracies: *constant* and *relative*. For example, a thermometer that is calibrated to $\pm 0.5°$ measures all values on its scale to within one degree of accuracy regardless of the value of the measured quantity. However, the expected error also can depend on the magnitude of the measured quantity. For example, the accuracy of a voltmeter might be $\pm 0.5\%$. That is, if the voltage measures 10 volts, we can expect this measurement to be accurate within $\pm$ 0.5% of 10 volts, or 0.1 volts. However, if the voltmeter displays a value of 100 volts, then this measurement is only accurate within 1.0 volts. In domains involving the first type of inaccuracy (constant), it seems appropriate to evaluate predictive ability in terms of absolute error; in domains involving the second type of inaccuracy (relative), one should instead measure the percentage error.

To study the effect of noise due to constant inaccuracy, we gave IDS the same data as in the previous experiment: randomly generated values of six numeric attributes (a, b, c, d, e, x) that obeyed the relation $x = 2.0 \times \frac{abc}{d^2} + 30.0$. We then corrupted each value of x by replacing its value with a sample taken from a normal distribution with a mean of x and a standard deviation of σ. In each experimental condition we used a different value of σ. We did not corrupt the values of the other attributes even though it is unrealistic to assume real-word noise occurs only in one attribute. However, this allows for a far better comparison of different levels of noise in the data, because the amount of noise is independent of the function. As in the previous experiment, we used an independent test set of 30 noise-free instances to measure the average (absolute) accuracy of prediction over 30 different runs. The value of the correlation threshold for this and all other experiments was 0.99.

Figure 3b shows the learning curves for IDS as the noise levels vary. The learning rate does not differ substantially for different amounts of noise. In most runs, it took the system less than 25 instances to find the correct term regardless of the noise level. However, the noise level influences the estimates for the slope and the intercept, which determine the accuracy after the correct term is found. We should note that when

the noise level was at $\sigma = 250.0$, IDS often could not find a law using the training set. Given only 75 data points, even the correlation between x and the correct numeric term was often less than 0.99 because of the noise in the data. However, if we set the correlation threshold to a lower value or used more training instances, then the system found the correct term in all runs.

In dealing with constant inaccuracy, IDS uses a least squares regression algorithm to estimate the slope and intercept, which minimizes the sum of the squares of the deviations over all data points and thus also minimizes the variance. However, relative inaccuracy causes the expected error to increase as the magnitude of the attribute becomes larger, so that deviations are not equally distributed over the range of values. Thus, if the expected error is proportional to the magnitude of the value and the values are spread over a wide range, then the basic least squares method will essentially ignore the small values in its estimation and produce poor estimates for the parameters in the linear relation.

In order to avoid this problem, IDS uses a variant of the *weighted least squares method* (Box, Hunter, & Hunter, 1978) to calculate the parameters of a linear relation if the expected error is proportional to the magnitude of the attribute. In such cases, this variant produces better estimates for the slope and intercept than the regular least squares method. Nordhausen (1989) reports an experiment similar to the one above, in which IDS uses this second method to discover laws with different levels of relative inaccuracy. As before, the learning rate (with *percentage* error as the dependent variable) decreased only slightly as the noise level increased, and in most runs IDS took less than 25 instances to find the correct term. Again the error rate affected the predictive accuracy once the correct term was found. Furthermore, the data with a 10% noise level proved too noisy in some cases, and the system failed to discover any law in five out of the 30 runs.

Whether the noise results from constant or relative inaccuracy of the measurement instruments, IDS' numeric discovery component proved to be robust. Correlation analysis seems to be a good heuristic to guide the search through IDS' space of possible numeric terms, even in the presence of noise. Once the system finds the correct term, the noise affects the estimation of the parameters of the linear relation, but as the system receives more instances, the regression algorithm produces estimates of increasing accuracy. Moreover, we used the same correlation threshold value for all conditions in the two experiments reported in this section. These experiments demonstrate that, even though the value of the correlation threshold affects the behavior of IDS, the system is not overly sensitive to this parameter and that the same correlation threshold value can be used for data covering a wide range of noise.

3.3 The Influence of Irrelevant Attributes

In the real world, one cannot always determine relevance of an attribute a priori. In the previous experiments, we always included one irrelevant attribute in the data. In another experiment, we investigated how irrelevant attributes affect IDS' learning rate. We again gave the system randomly generated data of numeric attributes obeying the function $x = 2.0 \times \frac{abc}{d^2} + 30.0$. We corrupted the values of x using noise due to constant inaccuracy (with $\sigma = 100.0$) as described in the previous section. However, in this experiment we varied the number of irrelevant attributes included in the data from zero to 20.

As before, we gave the system 30 different data sets for each experimental condition, and measured the absolute difference between the actual and predicted value. Figure 4a shows that the number of irrelevant attributes only increases the number of observations required to find the correct term. Even with 20 irrelevant attributes, IDS found the correct term within at most 24 instances. As the number of irrelevant attributes increases, the probability of an accidental correlation between x and a term involving irrelevant attributes becomes higher. However, given enough instances the numeric discovery component finds the correct term, so that irrelevant attributes affect only the learning rate. Once the system has identified the correct term, it ignores the irrelevant attributes, and the predictive accuracy becomes identical in all cases. Thus, the numeric learning component is robust even in the presence of many irrelevant variables.

3.4 The Influence of Function Complexity

Finally, in order to test the influence of function complexity, we ran the system on data obeying different laws in which we varied the number of variables and their degree. In this experiment, we randomly generated data for numeric attributes obeying one of three laws: $x = 2.0 \times ab + 30.0$, $x = 2.0 \times \frac{ab}{c} + 30.0$, or $x = 2.0 \times \frac{abc}{d^2} + 30.0$. These functions were chosen arbitrarily but increase in complexity. As before, we included one irrelevant attribute in each of the three data sets. The values for attributes a through d ranged from 10.0 to 100.0. Because of the different terms, the values of x ranged over a different interval for each function. Thus, we chose to introduce noise from relative inaccuracy (with a value of $\sigma = 0.75\%$) rather than constant inaccuracy, so the level of noise was comparable for each data set. Hence, we used percentage error rather than the absolute difference as the dependent measure in this experiment.

Figure 4b shows the learning curves for the three functions averaged over 30 random data sets, each containing 75 instances.[6] For the simplest function,

[6] The reason for the aberrations in the learning curves can be attributed to the runs in which the system replaces an incorrect term with an improved term that describes the processed training data better, but which actually does worse than the replaced term on the test data.

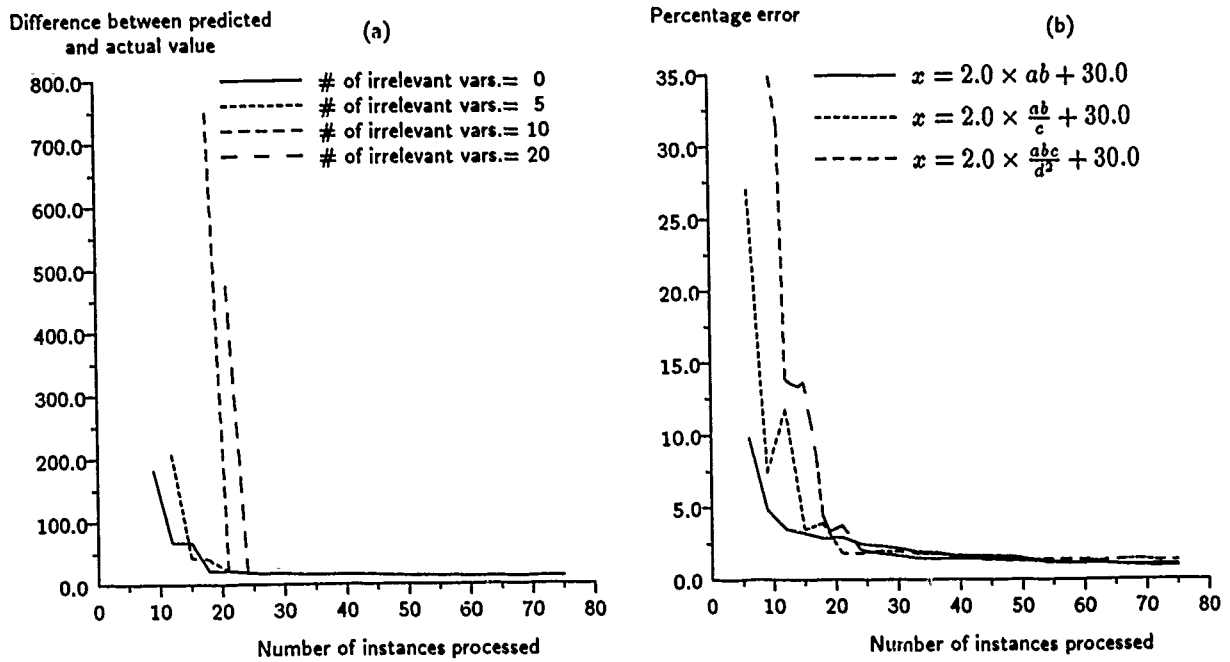


Figure 4. (a) The influence of irrelevant attributes on predictive ability.
(b) The influence of the function complexity on predictive ability.

IDS found the correct term rather quickly, requiring at most nine instances. There are two reasons the system needs more instances to discover the term as the complexity of the function increases. As we showed in the previous experiment, when the number of observed attributes increases, then the probability for accidental correlations between x and an incorrect term also increases. Furthermore, as the function becomes more complex, the path for the correct term through the space of possible numeric terms increases in length. However, once IDS finds the correct term, the predictive accuracy is similar regardless of the function's complexity. This suggests that function complexity only affects the number of instances required to find the correct term, but not the predictive accuracy once this term is found.

4 Discussion

In closing, we should attempt to draw some conclusions from our experiences with IDS. Below we discuss some strengths and limitations of the numeric algorithm, followed by the advances we believe our research has made over earlier approaches to discovery.

4.1 Strengths and Limitations

In the previous section, we reported a number of experiments with IDS' numeric discovery component. The studies produced encouraging results on the use of correlation analysis as a heuristic to guide the search for constant numeric terms. We saw that IDS is not overly sensitive to the setting of the correlation threshold, and that it can use a single value to discover relations of data with varying degrees of noise. In addition, the system tolerates noise that results from both constant and relative inaccuracy of measurement instruments. One must adjust the regression algorithm for the two kinds of noise, but it seems reasonable to assume one knows the kind of error a measurement instrument produces. Furthermore, we ran experiments that showed IDS tolerates irrelevant variables well. In these studies, the learning rate was hardly affected even when a large percentage of the observed attributes were irrelevant. Finally, we showed that the learning rate decreases only slowly as the complexity of the function describing the data increases.

However, one should treat these results with caution, because there are some well-known problems with correlation and regression analysis (Boz et al., 1978). For example, if a numeric attribute has only a limited range, then correlation analysis will fail to detect a law including that attribute. Furthermore, although we examined the effects of noise, irrelevant terms, and complexity in isolation, we did not consider the possibility of interactions among these factors. It seems plausible that IDS would encounter difficulty in noisy domains with complex functions and many irrelevant terms. Moreover, since the numeric discovery process is embedded within IDS' mechanisms for taxonomy formation and qualitative discovery, it relies on the successful operation of these mechanisms to obtain useful results. The experiments described above tested the numeric component in isolation, rather than in the context of the complete system. Finally, we have yet to determine the algorithm's behavior on real-world data. Future studies should address all of these issues.

4.2 Advances Over Previous Work

IDS finds numeric laws similar in form to those produced by BACON (Langley et al., 1983), ABACUS (Falkenhainer & Michalski, 1986), and FAHRENHEIT (Langley & Zytkow, 1989). Moreover, they employ similar methods to control their search for useful numeric terms, using correlation-like techniques to focus attention. However, these systems differ in the details of their search control. BACON and FAHRENHEIT employ a heuristic form of depth-first search, focusing on more recently defined terms. ABACUS creates a proportionality graph to determine promising combinations of terms, then uses a modified beam search to find the best combinations. IDS carries out a similar but less sophisticated beam search, relying on correlation analysis to guide its steps through the space of numeric terms. With respect to robustness, IDS provides a clear advance over BACON and FAHRENHEIT, which used simple 'trend detectors' rather than correlation, giving them only limited abilities for handling noise and irrelevant terms. The relation to ABACUS on this front is less clear, but future studies should compare the behavior of these algorithms.

Another difference between IDS and previous numeric discovery systems is its incremental approach and its reuse of existing terms. The current implementation requires reuse of all previous instances stored at a given node or link, but a version that computed the correlation scores incrementally would be considerably more efficient than a nonincremental version for domains in which observations are made a few at a time. Moreover, the reuse of existing terms should have a significant impact in domains involving many variables.

However, the most significant difference between IDS and earlier systems concerns the statement of conditions on the discovered laws. BACON, FAHRENHEIT, and ABACUS all include conditional statements on their numeric relations, but these statements contain little information about the structure or physical situation in which the laws occur. Even as simple a relation as the ideal gas law actually involves a set of structurally-related objects that change over time, which IDS can represent using a single abstract qualitative state. More complex laws involve sequences of qualitative states that together specify qualitative laws, as in the laws of fluid flow and in the generalization that acids react with alkalis to produce salts. In each case, the system annotates these qualitative summaries with numeric relations, providing detail beyond that usually included in qualitative representations.

Simultaneously, IDS' taxonomy of qualitative states and its abstract successor links specify a *context* for the quantitative laws it discovers. These constrain both the situations in which the numeric laws are applied and the search for relations among numeric quantities, since a law makes sense only when embedded within some qualitative description. Moreover, as Nordhausen (1989) describes in detail, the explicit representation of state transitions and time support the discovery of terms and laws that other systems cannot handle. Thus, IDS defines the boiling point of a substance as an intrinsic property associated with a law describing the transition between states. Also, by treating the duration of a qualitative state as an explicit quantity, the system can discover the law governing radioactive decay, in which the 'rate of reaction' depends on the amount of material. In summary, IDS' augmented representation allows some significant advances over previous approaches to numeric discovery.

Acknowledgements

The ideas in this paper have benefited from discussions with many of our colleagues in machine discovery, including Randy Jones, Don Rose, Deepak Kulkarni, Jan Zytkow, and Brian Falkenhainer. In addition, suggestions by Deepak Kulkarni, Kate McKusick, Kevin Thompson, and the reviewers helped us improve the presentation. This research was supported in part by Contract N00014-84-K-0345 from the Computer Science Division, Office of Naval Research.

References

Boz, G. E. P., Hunter, W. G., & Hunter, J. S. (1978). *Statistics for experimenters.* New York, NY: John Wiley.

Falkenhainer, B. C., & Michalski, R. S. (1986). Integrating quantitative and qualitative discovery: The ABACUS system. *Machine Learning, 1*, 367–422.

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

Forbus, K. D. (1985). Qualitative process theory. In D. G. Bobrow (Ed.), *Qualitative reasoning about physical systems.* Cambridge, MA: MIT Press.

Freund, J. E., & Walpole, R. E. (1980). *Mathematical statistics.* Englewood Cliffs, NJ: Prentice-Hall.

Langley, P., Bradshaw, G. L., & Simon, H. A. (1983). Rediscovering chemistry with the BACON system. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 1). San Mateo, CA: Morgan Kaufmann.

Langley, P., & Zytkow, J. M. (1989). Data-driven approaches to empirical discovery. *Artificial Intelligence, 40*, 283–312.

Lebowitz, M. (1987). Experiments with incremental concept formation: UNIMEM. *Machine Learning, 2*, 103–138.

Lenat, D. B. (1978). The ubiquity of discovery. *Artificial Intelligence, 9*, 257–285.

Nordhausen, B. (1989). *An integrated framework for empirical discovery.* Doctoral dissertation, Department of Information and Computer Science, University of California, Irvine.

More Results on the Complexity of Knowledge Base Refinement: Belief Networks

Marco Valtorta
Department of Computer Science
University of South Carolina
Columbia, South Carolina 29208, U.S.A.
(803)777-4641
mgv@cs.scarolina.edu (...!usceast!mgv)

Abstract

The refinement of knowledge bases is an important activity in the expert system lifecycle. Belief networks have been proposed and studied as an alternative to rules in expert system knowledge bases. The problems of synthesis and refinement of belief networks arising from the development of a large belief-network-based knowledge-based system, are presented and formally analyzed. We prove that, for very simple Dempster-Shafer networks (trees), defining parameter values (synthesis) is NP-Complete. Additional results that are given without proof include the computational intractability of refining expert-estimated values (refinement), even when we settle for approximate values or demand agreement on only a certain percentage of cases. The potential impact of these results on the practice of expert system construction is discussed.

1. Introduction

Knowledge base refinement is concerned with modifications to a knowledge base that increase its breadth and accuracy.

Most published work in knowledge base refinement has addressed truth-functional rule bases. From the point of view of knowledge-base refinement, truth-functional (or extensional) and non-truth-functional (or intensional) systems are distinctly different [Ruspini, 1982; Pearl, 1988]. A rule-based expert system is *truth-functional* if the belief associated with a proposition in the system depends only on the belief in propositions that appear in the premise of rules that conclude the original proposition, with an obvious exception for the propositions that are not concluded by any rule. In his contribution to the knowledge-base refinement track of last year's International Workshop on Machine Learning, Valtorta [1989c] addressed the computational complexity of truth-functional rule base refinement.

We now report new results that extends the complexity analysis of knowledge base refinement to systems that are not truth-functional, viz. belief networks.

Belief networks are gaining popularity as a formalism for implementing knowledge bases for expert systems. With respect to the more common MYCIN-style rule bases, belief networks overcome the problems arising from a truth-functional approach to evidence propagation, by adopting a model-based (or intensional) approach, as explained, e.g., in [Pearl, 1988, chapter 1]. The use of a sounder approach to handling uncertainty has its drawback in the worst-case inefficiency of belief computation in belief networks [Cooper, 1988; Provan, 1989].

A key feature of belief networks is their use of numerical parameters. These parameters are probability masses in Dempster-Shafer networks (see [Smets, 1988] for a brief introduction) and conditional probabilities (or related parameters, such as likelihood ratios) in Bayesian networks (see [Pearl, 1988] for the canonical treatment]). These numerical parameters can be estimated using statistical techniques. However, this may be practically unfeasible, and developers must resort to knowledge engineering techniques, as documented in the construction of MUNIN.

There are only very few large applications of belief networks. Of these, probably the best known is MUNIN [Andreassen et al, 1987]. As of mid-June 1989, MUNIN had grown to a network of approximately 1000 nodes.[1] "Estimating the 270 conditional probabilities [in the MUNIN belief network of 1987] (...) would require at least 10000 cases. Instead of relying on this empirical approach, we have tried to rely as much as possible on 'deep knowledge', using an understanding of patophysiological processes as expressed in medical textbooks and papers" [Andreassen et al., 1987, p.369]. In any case, whatever

[1]Steen Andreassen, personal communication. MUNIN has been developed in the context of ESPRIT project 599.

the source of the numerical parameters, they need to be refined. In MUNIN, "Discrepancies between the network and the medical experts lead to revision of the model parameters (...). Occasionally, it may even be necessary to modify the structure of the network, adding or deleting states or nodes" [Andreassen et al., 1987, p.370]. Discrepancies between the network and the expert are uncovered by comparing the belief assigned to particular variables in particular nodes by the expert and by the network, when the same evidence is presented to the expert and to the network. In MUNIN, the evidence typically describes the findings for a patient, while the particular variables correspond to possible diagnoses.

This paper addresses the problem of synthesis and refinement of numerical parameters in Dempster-Shafer belief networks from an algorithmic standpoint. The author has obtained analogous results for Bayesian networks [Valtorta and Loveland, 1989]. Section 2 formalizes the problem already described in this introduction, by defining, among others, the notion of case. In section 3 we show that the synthesis of masses in Dempster-Shafer networks from cases is NP-Complete. Additional results, including the proof that the refinement of masses in Dempster-Shafer networks from cases is NP-Complete, and some settings involving approximations, are presented (without proof) in section 4. Section 5 discusses related work. Section 6 concludes the paper with an assessment of results.

The proofs that are not in the paper and much additional material can be found in [Valtorta and Loveland, 1989], which has been submitted for journal publication.

2. Formalizing Mass Refinement

Without loss of generality, since we are after lower bound results, we consider Dempster-Shafer networks (from now on, *DS-nets)* in the form of a tree (and call them, simply, *DS-trees).* There are a considerable number of different versions of DS-trees in the literature, all related in rather straightforward ways. In this paper, alternating Markov trees are used. Our definition is adapted from [Mellouli, 1988, pp.66,85]. A *(qualitative) Markov tree* of variables in a set S is a tree T = (N, E), such that N is a subset of the power set of S (i.e., the nodes of the tree are subsets of S) and such that the intersection of two nodes n_1 and n_2 is contained in node n_0 if n_0 lies between n_1 and n_2 in some branch of T. A Markov tree T = (N,E) is *alternating* if every node in the tree is either contained in all its neighbors or contains all its neighbors. (For example, the tree in Figure 1 is an alternating Markov tree for variables in the set S = $\{c_1, c_2, ..., c_n, u\}$.)

For simplicity (and again, without loss of generality), each of the variables in the DS-tree will be assumed to be two-valued. Call the two values 0 and 1. We will express the mass assigned to a subset of the frame of discernment[2] of variable a as $m(a=v_i)$, where $v_i = 0,1$, or as $m(l_a)$, where $l_a = a, \neg a$. The mass assigned to the frame of discernment of variable a will be indicated as $m(\Theta(a))$ or, in case there is no ambiguity, simply as $m(\Theta)$. For joint variables, we will indicate a subset of the frame of discernment by a propositional formula. For example, consider the joint variable (a,b) whose frame of discernment is $\Theta((a,b))$ = {{a=0,b=0}, {a=0,b=1}, {a=1,b=0}, {a=1,b=1}}. The mass assigned to the subset {{a=0,b=0}, {a=0, b=1}, {a=1, b=1}} will be indicated as $m(\neg a \vee b)$, or as $m(a \Rightarrow b)$. The mass assigned to the frame of discernment of the joint variable (a,b) will be indicated as $m(\Theta((a,b)))$ or, in case there is no ambiguity, simply as $m(\Theta)$. Following Pearl [1988, p. 418], we call a subset of the frame of discernment such as $(\neg a \vee b)$ a *compatibility relation.* (Intuitively, $m(\neg a \vee b)$ quantifies the constraint that a is not compatible with $\neg b$.)

A *DS-presentation* is defined as a triple consisting of a DS-tree, a set of compatibility relations, and an assignment of masses to some of the compatibility relations and their frames of discernment. Figure 1 shows a DS-presentation when values $s_1, s_2, ..., s_n$ are fixed. DS-presentations will be considered as realizing a function from the vector of masses assigned to the leaf nodes of a DS-tree to a belief (simply a mass for the nets that we consider in this paper) for the root node (via a process involving Dempster's rule and summarized later). A point in the graph of the function will be called a *case.*[3] We now describe how this models the situation described by Andreassen et al. [1987] and summarized in the introduction. The output part of a case describes the desired "answer" of the DS-tree when "queried" with the evidence encoded as the input part of the same case. Typically, there will be a discrepancy *(output error)* between the value of the belief as computed by the tree and the belief given as output part of the case. This discrepancy must be eliminated in order for the DS-tree to work correctly.

Before addressing the task of refinement, we address in the next section the more basic task of

[2]See, e.g., [Smets, 1988] or [Gordon and Shortliffe, 1985] for the definition. Informally, the frame of discernment for a set S of variables (where all variables have a discrete range of values) is the Cartesian product of the set of variable-value pairs for all variables in S.

[3]T_1 through T_4 of Figure A (in the Appendix) are examples of cases. Each case is an input-output pair.

parameter instantiation (assigning values to the parameters). We call this the *synthesis* task.

3. Initial Mass Synthesis Is NP-Hard

The problem considered in this section is a problem of synthesis, rather than a problem of refinement of masses in DS-trees.

Problem name: Mass Synthesis (MS).

Problem instance: A DS-tree and associated compatibility relations as given in Figure 1; a set of cases.

Question: Is there an assignment of values to $s_1,...,s_n$, such that the function realized by the DS-tree satisfies the cases?

Theorem 1

MS is NP-Complete.

The proof is given in the Appendix.

4. Mass Refinement Is NP-Hard

Problem name: Mass Refinement, Search Version (MRS).

Problem instance: A DS-tree and associated compatibility relations as given in Figure 1; an assignment of values[4] for $s_1,...,s_n$; a positive constant e; a set of cases.

Question: Find an assignment of values to $s_1,...,s_n$ each of which is at most e away from the given assignment and such that the function realized by the DS-tree satisfies the cases.

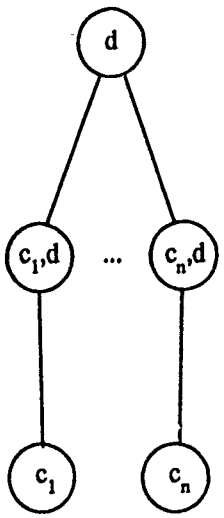
MRS is NP-Hard if the following decision problem is:

Problem name: Mass Refinement (MR).

Problem instance: A DS-tree and associated compatibility relations as given in Figure 1; an assignment of values for $s_1,...,s_n$; a positive constant e; a set of cases.

Question: Is there an assignment of values to $s_1,...,s_n$ each of which is at most e away from the given

[4]These values may be expert-given or otherwise estimated.



$m_1(c_1 \Rightarrow d) = s_1,\ m_1(\Theta) = 1 - s_1$
$m_2(c_2 \Rightarrow d) = s_2,\ m_2(\Theta) = 1 - s_2$
. . . .
$m_n(c_n \Rightarrow d) = s_n,\ m_n(\Theta) = 1 - s_n$

Figure 1 DS-tree and associated compatibility relations for problem MS.

assignment and such that the function realized by the DS-tree satisfies the cases?

Theorem 2

MR is NP-Complete for any fixed e.

We now present three problems that arise from attempts to define meaningful approximate solutions to the refinement of numerical parameters.

Problem name: Approximate Mass Synthesis (AMS).

Problem instance: As for problem MS in section 3.

Question: Find an assignment to $s_1,...,s_n$ strictly within 0.5 of the correct assignment, where the correct assignment is the assignment such that the function realized by the DS-tree satisfies the cases.

Theorem 3

AMS is NP-Hard.

Problem name: Mass Synthesis, Bounded output Error (MSBE).

Problem instance: A DS-tree and associated compatibility relations, as given in Figure 1; a set of cases; a constant e.

Question: Is there an assignment of values to $s_1,...,s_n$ such that the maximum output error on the cases is less than or equal to e?

Theorem 4

MSBE is NP-Hard for all e<=d<.5.

Remark

To prove Theorem 4, we need some technical conditions on the possible values of $s_1,...,s_n$. For a different kind of refinement involving bounded output error, see the discussion at the beginning of the next section.

Problem name: Noisy Mass Refinement (NMR).

Problem instance: A DS-tree and associated compatibility relations as given in Figure 1; an assignment of values for $s_1,...,s_n$; a positive constant e; a positive constant k less than 100; a set of cases.

Question: Is there an assignment of values to $s_1,...,s_n$ each of which is at most e away from the given ones and such that the function satisfies k% of the cases?

Theorem 5

NMR is NP-Hard.

5. Related Work

In applications (such as diagnosis) in which the *task* (as defined, e.g., in [Breuker et al., 1987]) of the expert system using the belief network is to classify, the user is concerned with the relative ranking (rather than the exact values) of beliefs associated to the terminal nodes[5] of a DS-net or a Ba-net. Cooper [1988, sections 5.4 and 5.5] and Valtorta [1989a, 1989c] provide additional motivation and techniques. The corresponding refinement problems are NP-Hard.

The introduction points to some of the work on rule bases, a common form of knowledge base. Synthesis of numerical parameters in rule bases is somewhat analogous to training in neural networks. Indeed there are apparently similar results of intractability. (Cf. [Judd, 1987, 1988; Blum and Rivest, 1988; Lin and Vitter, 1989].) However, the functions used in neural networks to process weights are different from those used in rule bases or belief networks. (Cf. [Fu, 1988; Valtorta, 1989a].) As done in much of the neural network training work, we assume that the structure of the net is fixed. Typically, it is given by the expert. The fixed network assumption is therefore more appropriate in belief network refinement than in neural network training. We conjecture that refinement of belief networks remains NP-Hard when limited changes to the structure of the network are permitted.

6. Conclusion

The networks used in the problem instance of MS and MR are trees. It is well known (e.g., [Kong, 1986; Pearl, 1988; Lauritzen and Spiegelhalter, 1988; Shenoy and Shafer, 1988]) that the computation of beliefs in trees[6] is tractable. Therefore, a developer could be faced with the unpleasant situation in which the belief network is nicely structured for efficient *computation* of beliefs, but *refinement* is extremely difficult.

Synthesis of numerical parameters in knowledge bases is a kind of refinement of knowledge bases: the numerical estimates of the parameters are not available, while the structure of the knowledge base is. The structure of the knowledge base can be determined by answering relatively simple questions about independence of events. Therefore, the knowledge engineer should believe more strongly in the (qualitative) network structure than in the values of the numerical parameters, and it is natural to use an expert to obtain the knowledge structure and initial guesses. These considerations suggest a methodology for the construction of knowledge-based systems that use belief networks. At the heart of this methodology is a *propose and fit* cycle. In this cycle, after interviewing an

[5] when suitably defined, in the obvious way.

[6] The same is true for some kinds of graphs that are not trees.

expert, the structure of a belief network is proposed. The network parameters are set or adjusted to fit selected test cases. If parameters can be set to fit the cases, the development is complete. If they cannot, the designer needs to consult the expert further until a new (qualitative) network structure is proposed. Another attempt is made to set the parameters to fit the cases, and so on. Our results indicate that it is hard to automate the "fit" step of this methodology.

As an example of alternative methods for validation and refinement, consider the use of a type of oracles: If an expert is available after construction of the knowledge base, the expert could be used as an oracle to facilitate knowledge base refinement. In this mode, the expert would be queried with specially focused questions allowing the synthesis or refinement of specific masses or likelihoods. A result by Valtorta concerning rule bases [1987, chapter 4 and chapter 7; 1989b] indicates that automatic synthesis or refinement in certain belief networks that are trees is doable in polynomial time and suggests that it is intractable for graphs. More remains to be done along this line of work.

Acknowledgements

We gratefully acknowledge many useful discussions with Donald Loveland of Duke University. The referees suggested several improvements.

References

Andreassen, S., M. Woldbye, B. Falck, S.K. Andersen. "MUNIN--A Causal Probabilistic Network for Interpretation of Electromiographic Findings." *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 366-372, 1987.

Blum, A. and R.L. Rivest. "Training a 3-Node Neural Network is NP-Complete." *Proceedings of the 1988 Workshop on Computational Learning Theory (COLT-88)*, 9-18

Breuker, J., B. Wielinga, M. van Someren, R. de Hoog, G. Schreiber, P. de Greef., B. Bredeweg, J. Wielemaker, J.-P. Billault, M. Davoodi, S. Hayward. "Model-Driven Knowledge Acquisition: Interpretation Models." Deliverable task A1, ESPRIT Project 1098, and Memo 87, VF Project Knowledge Acquisition in Formal Domains, 1987. (This report is available from the Department of Social Science Informatics, University of Amsterdam.)

Cooper, G.F. "Probabilistic Inference Using Belief Networks is NP-Hard." Stanford University Knowledge Systems Laboratory Memo KSL-82-27, May 1987 (revised July 1988).

Fu, L. "Truth Maintenance Under Uncertainty." *Proceedings of the Fourth Workshop on Uncertainty in Artificial Intelligence*, 119-126, 1988.

Garey, M.R. and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman, 1989.

Gordon, J. and E.H. Shortliffe. "A Method for Managing Evidential Reasoning in a Hierarchical Hypothesis Space." *Artificial Intelligence*, 26 (1985), 323-357.

Judd, S. "Complexity of Connectionist Learning with Various Node Functions." Technical Report 87-60, University of Massachusetts at Amherst, July 1987.

Judd, S. "Learning in Neural Networks." *Proceedings of the 1988 Workshop on Computational Learning Theory (COLT-88)*, 2-8.

Kong, C.T.A. "Multivariate Belief Functions and Graphical Models." Ph.D. Dissertation, Department of Statistics, Harvard University, 1986. (Available as Research Report S-107, Department of Statistics, Harvard University.)

Lauritzen, S.L. and D.J. Spiegelhalter. "Local Computations with Probabilities on Graphical Structures and their Applications to Expert Systems." *Journal of the Royal Statistical Society, Series B (Methodological)*, 50 (1988), 2, 157-224.

Lin, J.-H. and J.S. Vitter. "Complexity Issues in Learning by Neural Nets." *Proceedings of the Second Annual Workshop on Computational Learning Theory (COLT-89)*, 118-133, 1989.

Mellouli, K. "On the Propagation of Beliefs in Networks Using the Dempster-Shafer Theory of Evidence." Ph.D. Dissertation and Working Paper No. 196, School of Business, University of Kansas, April 1988.

Pearl, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, California: Morgan-Kaufmann, 1988.

Provan, G.M. "A Logic-Based Analysis of Dempster-Shafer Theory." Report, Department of Computer Science, University of British Columbia, October, 1989.

Ruspini, E.H. "Possibility Theory Approaches for Advanced Information Systems." *Computer*, 15, 9 (September 1982), 83-91.

Shafer, G., P.P. Shenoy, and K. Mellouli. "Propagating Belief Functions in Qualitative Markov Trees." *International Journal of Approximate Reasoning*, 1987, 349-400.

Shenoy, P.P. and G. Shafer. "Propagating Belief Functions with Local Computations." *IEEE Expert*, 1, 3 (Fall 1986), 43-52.

Shenoy, P.P. and G. Shafer. "An Axiomatic Framework for Bayesian and Belief-function Propagation" *Proceedings of the Fourth Workshop on Uncertainty in Artificial Intelligence*, 307-314, 1988.

Smets, P. "Belief Functions." Chapter 9 in: Smets,

P., E.H. Mamdani, D. Dubois, and H. Prade. *Non-Standard Logics for Automated Reasoning.* London: Academic Press, 1988.

Valtorta, M. "Automating Rule Strengths in Expert Systems." Ph.D. Dissertation, Department of Computer Science, Duke University, April 1987. (Also: Technical Report CS-1987-15, Department of Computer Science, Duke University and available as number ADG87-25869 from University Microfilm International.)

Valtorta, M. (a) "Some Results on the Complexity of Knowledge Base Refinement" Technical Report TR89004, Department of Computer Science, University of South Carolina, April 1989 (Revised version accepted for publication in the *International Journal of Approximate Reasoning.)*

Valtorta, M. (b) "Some Results on Knowledge Base Refinement with an Oracle." Technical Report TR89005, Department of Computer Science, University of South Carolina, April 1989.

Valtorta, M. (c) "Some Results on the Complexity of Knowledge-base Refinement." *Proceedings of the Sixth International Workshop on Machine Learning*, 323-331.

Valtorta, M. and D.W. Loveland. "On the Complexity of Belief Network Synthesis and Refinement." Technical Report TR89011, Department of Computer Science, University of South Carolina, November 1989 (submitted for journal publication).

Appendix: proof of Theorem 1

One in Three Satisfiability (OTS) [Garey and Johnson, 1979, p.259] will be transformed into MS. The variant in which no clause in the formula contains a negative literal will be used. The generic OTS instance is a formula in 3-conjunctive normal form, with no negated variables. The question is whether there is a model for the expression such that each clause has exactly one true variable.

Given a formula E in monotone 3-conjunctive normal form, the following algorithm produces in time polynomial in the size of E an instance of MS such that the Question has answer yes if and only if E has a model in which only one variable per clause is true.

Let n be the number of distinct propositional variables in E, m be the number of clauses in E. (n and m can be obtained in polynomial time from any reasonable encoding of E.) (Name the variables $x_1,...,x_n$ for convenience.)

The number of leaves in the DS-tree of the corresponding MS-instance is n. The number of cases in the corresponding MS-instance is 2m.

There are 2 cases for each clause in E. The cases are defined as follows. Let a and b be a pair of numbers such that $0<a<b<1$. Let a generic clause contain the variables x_i, x_j, x_k. The input part of the first case for each clause has $m(c_i) = m(c_j) = m(c_k) = a$ and 0 everywhere else. The output part of the first case for each clause is a. To obtain the second case for this clause, substitute b for a.

The reader can easily verify that the algorithm just given runs in time polynomial in the size of E.

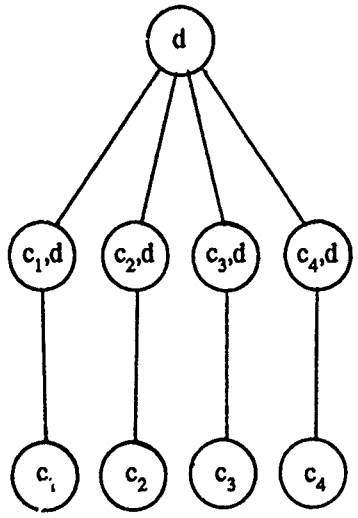
As an example, Figure A shows the instance of MS corresponding to $E = (x_1 v x_2 v x_3) \& (x_1 v x_2 v x_3)$. In the Figure, T_1 and T_2 correspond to the first clause in E, while T_3 and T_4 correspond to the second clause in E.

In order to prove that an instance of MS built according to the algorithm just given is a yes-instance if and only if the corresponding instance of OTS is a yes-instance, the following fact is useful.

Let [p+] denote the *probabilistic sum* operator, defined as a [p+] $b = a + b - ab$. It is easy to show, on the basis of an observation by Gordon and Shortliffe [1985, section 3.3], that, indicating the belief in d as Bel(d),

$Bel(d) = m(c_1)*s_1$ [p+] . . . [p+] $m(c_n)*s_n$.[7]

[7]Each m should have a different subscript, which is dropped for readability.



$m_1(c_1 \Rightarrow d) = s_1$, $m_1(\Theta) = 1 - s_1$

....

$m_4(c_4 \Rightarrow d) = s_4$, $m_4(\Theta) = 1 - s_4$

T_1: ((a,a,a,0),a)
T_2: ((b,b,b,0),b)

T_3: ((a,a,0,a),a)
T_4: ((b,b,0,b),b)

Figure A Instance of MS corresponding to $(x_1 v x_2 v x_3)$ & $(x_1 v x_2 v x_4)$.

The "if part" is simpler. If variable x_i in the model for E is true, set s_i to 1. Otherwise, set it to 0. This insures that exactly one of the masses corresponding to each case is 1 and the other two are 0. Therefore, the computed Bel is equal to the mass. Therefore, each case is satisfied.

The "only if" part is proved now. Assume that we have a yes-instance of MS. It will be shown that, in order for an instance of MS to be a yes-instance, it must be that exactly one of the s_i corresponding to each case is 1 and the other two are 0. By assigning true to variable corresponding to this single s_i, a model for E is obtained that satisfies the "one in three" condition. Consider a generic pair of cases corresponding to a clause in E. We show, by algebraic manipulation, that this pair is satisfied if and only if exactly one of the three s_i corresponding to the cases is 1 and the other two are 0. Call the strengths x, y, and z. The pair of cases is satisfied if and or llowing system has a solution:

ax[p+]ay[p+]az = a
bx[p+]by[p+]bz = b,
i.e., after carrying out the probabilistic sums and dividing each side by a,

$x + y - axy + z - axz - ayz + a^2xyz = 1$
$x + y - bxy + z - bxz - byz + b^2xyz = 1.$

If any two of x, y, and z have value 0, the system has a solution if and only if the other variable has value 1.

To show that the system has no solution if only one of the three variables is 0, subtract the second from the first equation side by side and divide by (b-a):

$xy + xz + yz = (b+a)xyz.$

This equation has no solution if only one of the three variables is 0.

The only case left is that in which the three variables are all positive (and, of course, no greater than 1). In this case, each of the products xy, xz, and yz is greater than or equal to xyz:
$xy + xz + yz > 2xyz > (b+a)xyz$, and therefore it is impossible that
$xy + xz + yz = (b+a)xyz$.

It has been shown that MS is NP-Hard. In order to complete the proof that MS is NP-Complete, it remains to show that MS is in NP. A non-deterministic program to solve MS has a loop whose body assigns (non-deterministically) a value to each mass and tests whether for that assignment the function realized by the DS-tree satisfies all cases. Since the test can be performed in deterministic polynomial time, the whole program runs in non-deterministic polynomial time.

(End of proof of Theorem 1)

Remark.

Note that we have not specified the possible values for $s_1,...,s_n$ in the statement of problem MS. The proof of NP-Hardness shows that MS is NP-Complete even when the values are restricted to be in {0,1}. If there are more than a constant number of different values, MS remains NP-Hard, but we cannot show it to be NP-Complete. Similarly, we have not specified the possible values for the input part of the cases. The proof shows that MS is NP-Hard even when the values are restricted to a and b, $0<a<b<1$. Finally, MS is NP-Complete even if both input and s_i values are restricted as just outlined at the same time.

INDEX