Proceedings of the 2ND KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff Conference Centre
Banff, Alberta, Canada
October 18-23, 1987

Sponsored by the
American Association for
Artificial Intelligence

Workshop Co-chairs:
John H. Boose, Boeing Computer Services
Brian R. Gaines, University of Calgary

Proceedings of the
2ND KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff Conference Centre, Banff, Alberta, Canada
October 19-23, 1987

Sponsored by the American Association for Artificial Intelligence

Workshop Co-Chairs:
John H. Boose, Boeing Advanced Technology Center
Brian R. Gaines, University of Calgary

These proceedings are being reissued for historical research purposes. Many of the workshop papers were further developed as journal publications and these constitute the primary scholarly record of the research.

2ND KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Sponsored by the American Association for Artificial Intelligence
Banff, Canada, October 19-23, 1987

CONTENTS

Program iii

List of Participants and Authors vi

Technical Papers (listed alphabetically by first author)

Protos: An Exemplar-Based Learning Apprentice 1-0
E. Ray Bareiss, Bruce W. Porter, Craig C. Wier, University of Texas

Uses of Repertory Grid-Centered Knowledge Acquisition Tools for Knowledge-Based Systems 2-0
John H. Boose, Boeing Advanced Technology Center

Decision Analytic Techniques for Knowledge Acquisition: Combining Situation and Preference Models Using Aquinas 3-0
Jeffrey M. Bradshaw, John H. Boose, Boeing Advanced Technology Center

What Kind of Information Processing is Intelligence? A Perspective on AI Paradigms and A Proposal 4-0
B. Chandrasekaran, The Ohio State University

Extensions to the CART Algorithm 5-0
Stuart L. Crawford, Advanced Decision Systems

MOLE: A Knowledge Acquisition Tool That Buries Certainty Factors 6-0
Larry Eshelman, Carnegie Mellon University

Knowledge Acquisition in a Secure Environment 7-0
Nancy E. Fredman, Planning Research Corporation

Advanced Expert System Support Environments 8-0
Brian R. Gaines, University of Calgary

Learning from Examples Using Information Theory 9-0
Rodney M. F. Goodman, Padhraic Smyth, California Institute of Technology

Acquiring Strategic Knowledge from Experts 10-0
Thomas Gruber, University of Massachusetts

ASTEK: A Multi-Paradigm Knowledge Acquisition Tool for Complex Structured Knowledge 11-0
Chris Jacobson, Michael J. Freiling (presenting), Tektronix, Inc.

Selecting Knowledge Acquisition Tools and Strategies Based on Application Characteristics 12-0
Catherine M. Kitto, John H. Boose, Boeing Advanced Technology Center

Knowledge Acquisition for Evaluation Systems 13-0
Georg Klinker (Carnegie Mellon University), Serge Genetet (Banque International de Placement), John McDermott (Digital Equipment Corporation)

Knowledge Acquisition as Knowledge Assimilation 14-0
Lawrence S. Lefkowitz, Victor R. Lesser, University of Massachusetts

Cognitive Primitives 15-0

Alain Rappaport, Neuron Data

The Problem of Intelligent Example Selection 16-0
Edwina L. Rissland, University of Massachusetts

A Formal Analysis of Machine Learning Systems for Knowledge Acquisition 17-0
Valerie L. Shalin, Edward J. Wisniewski, Keith R. Levi (Honeywell), Paul D. Scott (University of Michigan)

Validation of a Knowledge Support System 18-0
Mildred L. G. Shaw, J. Brian Woodward, University of Calgary

Refining Problem-Solving Knowledge in Repertory Grids Using a Consultation Mechanism 19-0
David B. Shema, John H. Boose, Boeing Advanced Technology Center

An Explanation-Based Approach to Knowledge-Base Acquisition 20-0
Kenneth Silvestro, Bowdoin College

Toward Automating Recognition of Differing Problem-Solving Demands 21-0
Jeffrey Stout (Carnegie Mellon University), Gilbert Caplain (CERMA-ENPC), Sandra Marcus (Boeing Advanced Technology Center), John McDermott (Digital Equipment Corporation)

A Knowledge Acquisition Environment for Scene Analysis 22-0
Deborah Tranowski, Advanced Decision Systems

Using Concept Learning for Knowledge Acquisition 23-0
Ian H. Witten, Bruce MacDonald, University of Calgary

Hitting a Moving Target (The Training of Knowledge Engineers) 24-0
J. Brian Woodward, University of Calgary

Design Goals for Sloppy Modeling Systems 25-0
Stefan Wrobel, Techn. Univ. Berlin

PROGRAM
SECOND AAAI KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Sponsored by the American Association for Artificial Intelligence
Banff, Alberta, Canada, October 18-23, 1987

Co-Chairmen
John H. Boose, Boeing Advanced Technology Center
Brian R. Gaines, University of Calgary

Program Committee
Jeffrey Bradshaw, Boeing Advanced Technology Center
B. Chandrasekaran, Ohio State University
Catherine Kitto, Boeing Boeing Advanced Technology Center
Sandra Marcus, Boeing Advanced Technology Center
John McDermott, Digital Equipment Corporation
Ryszard Michalski, University of Illinois
Mildred Shaw, University of Calgary

DAILY SCHEDULE

Workshop sessions will be held in the residence building. Continental breakfasts will be available before each session in the workshop area. There will be coffee and juice breaks each morning and afternoon, and lunch will be served from 12:00 - 1:00, also in the workshop area. Hot breakfasts and dinners will be available in a separate Centre building.

Talks will last 40 minutes - 20 minutes for presentation, and 20 minutes for discussion. Please keep strictly to the presentation time to allow ample time for discussion, and do not force us to resort to the use of bears.

SCHEDULE OF EVENTS

Sunday, October 18

2:00 - 9:00 Registration
7:00 Reception

Monday, October 19

Introduction
8:30 - 9:00 Welcome and Workshop Organization, John Boose and Brian Gaines

Overview and Workshop Theme: Integration Issues
9:00 - 9:40 Knowledge Acquisition Systems: Integration Issues
Brian Gaines, University of Calgary
9:40 - 10:20 Machine Learning and Knowledge Acquisition: Integration Issues
Ryszard Michalski, University of Illinois at Urbana

10:20 - 10:40 Break

10:40 - 11:20 Integration of Knowledge Acquisition Tools and Techniques: A Case Study
John Boose, Boeing Advanced Technology Center

Knowledge Acquisition Methods and Techniques - I
11:20 - 12:00 Acquiring Strategic Knowledge from Experts
Thomas Gruber, University of Massachusetts

12:00 - 1:00 Lunch

1:00 - 1:40 Design Goals for Sloppy Modeling Systems
Stefan Wrobel, Techn. Univ. Berlin
1:40 - 2:20 Knowledge Acquisition as Knowledge Assimilation
Lawrence S. Lefkowitz, Victor R. Lesser, University of Massachusetts

2:20 - 2:40 Break

Knowledge Acquisition Tools - I

2:40 - 3:20 MOLE: A Knowledge Acquisition Tool That Buries Certainty Factors
Larry Eshelman, Carnegie Mellon University

3:20 - 4:00 ASTEK: A Multi-Paradigm Knowledge Acquisition Tool for Complex Structured Knowledge
Chris Jacobson, Michael J. Freiling (presenting), Tektronix, Inc.

4:00 - 4:40 Knowledge Acquisition for Evaluation Systems
Georg Klinker (Carnegie Mellon University), Serge Genetet (Banque International de Placement), John McDermott (Digital Equipment Corporation)

Dinner

Tuesday, October 20

Knowledge Acquisition Methods and Techniques - II

9:00 - 9:40 What Kind of Information Processing is Intelligence? A Perspective on AI Paradigms and A Proposal
B. Chandrasekaran, The Ohio State University

Induction and Learning - I

9:40 - 10:20 A Formal Analysis of Machine Learning Systems for Knowledge Acquisition
Valerie L. Shalin, Edward J. Wisniewski, Keith R. Levi (Honeywell), Paul D. Scott (University of Michigan)

10:20 - 10:40 Break

10:40 - 11:20 Extensions to the CART Algorithm
Stuart L. Crawford, Advanced Decision Systems

11:20 - 12:00 The Problem of Intelligent Example Selection
Edwina L. Rissland, University of Massachusetts

12:00 - 1:00 Lunch

Knowledge Acquisition Tools - II

1:00 - 1:40 Refining Problem-Solving Knowledge in Repertory Grids Using a Consultation Mechanism
David B. Shema, John H. Boose, Boeing Advanced Technology Center

1:40 - 2:20 Toward Automating Recognition of Differing Problem-Solving Demands
Jeffrey Stout (Carnegie Mellon University), Gilbert Caplain (CERMA-ENPC), Sandra Marcus (Boeing Advanced Technology Center), John McDermott (Digital Equipment Corporation)

2:20 - 2:40 Break

Knowledge Acquisition Methods and Techniques - III

2:40 - 3:20 Cognitive Primitives
Alain Rappaport, Neuron Data

3:20 - 4:00 Validation of a Knowledge Support System
Mildred L. G. Shaw, J. Brian Woodward, University of Calgary

4:00 - 4:40 Hitting a Moving Target (The Training of Knowledge Engineers)
J. Brian Woodward, University of Calgary

Dinner

Wednesday, October 21

Knowledge Acquisition Tools - III

9:00 - 9:40 An Explanation-Based Approach to Knowledge-Base Acquisition
Kenneth Silvestro, Bowdoin College

9:40 - 10:20 Selecting Knowledge Acquisition Tools and Strategies Based on Application Characteristics
Catherine M. Kitto, John H. Boose, Boeing Advanced Technology Center

10:20 - 10:40 Break

Knowledge Acquisition Methods and Techniques - IV

10:40 - 11:20 A Knowledge Acquisition Environment for Scene Analysis
Deborah Tranowski, Advanced Decision Systems

11:20 - 12:00 Knowledge Acquisition in a Secure Environment
Nancy E. Fredman, Planning Research Corporation

Lunch, afternoon, evening - Local Atmosphere Checkup and Bear Patrol

Thursday, October 22

Induction and Learning - II

9:00 - 9:40 Learning from Examples Using Information Theory
Rodney M. F. Goodman, Padhraic Smyth, California Institute of Technology

9:40 - 10:20 Protos: An Exemplar-Based Learning Apprentice
E. Ray Bareiss, Bruce W. Porter, Craig C. Wier, University of Texas

10:20 - 10:40 Break

10:40 - 11:20 Using Concept Learning for Knowledge Acquisition
Ian H. Witten, Bruce MacDonald, University of Calgary

Knowledge Acquisition Methods and Techniques - V

11:20 - 12:00 Decision Analytic Techniques for Knowledge Acquisition: Combining Situation and Preference Models Using Aquinas
Jeffrey M. Bradshaw, John H. Boose, Boeing Advanced Technology Center

12:00 - 1:00 Lunch

Working Groups

Afternoon Methods, Induction and Learning, Tools

2:20 - 2:40 Break

Dinner

Friday, October 23

Working Group Presentations

9:00 - 9:40 Knowledge Acquisition Methods

9:40 - 10:20 Induction and Learning

10:20 - 10:40 Break

10:40 - 11:20 Knowledge Acquisition Tools

11:20 - 12:00 Synthesis and Summary - Brian Gaines

12:00 - 1:00 Farewell Lunch and Bear Souvenirs

LIST OF PARTICIPANTS AND AUTHORS

2ND AAAI KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff, Canada, October 19-23, 1987

Co-Chairs: John H. Boose and Brian R. Gaines

E. Ray Bareiss
Computer Science Department, University of Texas, Austin, TX 78712 USA

John H. Boose
Knowledge Systems Laboratory, Advanced Technology Center, Boeing Computer Services, P.O. Box 24346, Seattle, WA 98124 USA

Jeffrey M. Bradshaw
Knowledge Systems Laboratory, Advanced Technology Center, Boeing Computer Services, P.O. Box 24346, Seattle, WA 98124 USA

Gilbert Caplain
CERMA-ENPC, La Courtine B. P. 105, 93194 Noisy le Grand, France

B. Chandrasekaran
Laboratory for AI Research, The Ohio State University, Department of Computer & Informaito Science, Columbus, OH 43210 USA

Stuart L. Crawford
Advanced Decision Systems, 201 San Antonio Circle, Suite 286, Mountain View, CA 94040 USA

Larry Eshelman
Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 USA

Michael J. Freiling (presenting)
Tektronix Laboratories, Tektronix, Inc., Tektronix Industrial Park, P.O. Box 500, MS 50-662, Beaverton, Ore. 97077 USA

Nancy E. Fredman
Planning Research Corporation, 1500 Planning Research Drive, McLean, VA 22102 USA

Brian R. Gaines
Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4

Serge Genetet
Banque International de Placement, 108 Boulevard Hausmann, 75008 Paris, France

Rodney M. F. Goodman
Department of Electrical Engineering 116-81, California Institute of Technology, Pasadena, CA 91125 USA

Thomas Gruber
Experimental Knowledge Systems Laboratory, Computer and Information Science, University of Massachusetts, Amherst, MA 01003 USA

Chris Jacobson
Tektronix Laboratories, Tektronix, Inc., Tektronix Industrial Park, P.O. Box 500, MS 50-662, Beaverton, Ore. 97077 USA

Catherine M. Kitto
Knowledge Systems Laboratory, Advanced Technology Center, Boeing Computer Services, P.O. Box 24346, Seattle, WA 98124 USA

Georg Klinker
Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 USA

Lawrence S. Lefkowitz
Computer and Information Science Department, University of Massachusetts, Amherst, MA 01003 USA

Victor R. Lesser
Computer and Information Science Department, University of Massachusetts, Amherst, MA 01003 USA

Keith R. Levi
Honeywell
Systems and Research Center
3600 Technology Drive
Minneapolis, Minn. 55418 USA

Bruce MacDonald
Knowledge Sciences Institute, Department of Computer Science, The University of Calgary, 2500 University Drive NW, Calgary, Canada T2N 1N4

Sandra Marcus
Knowledge Systems Laboratory, Advanced Technology Center, Boeing Computer Services, PO Box 24346, Seattle, WA 98124 USA

John McDermott
Digital Equipment Corporation, HL02-3/CO7, 77 Reed Road, Hudson, MA 01749 USA

Ryszard S. Michalski
Dept. of Computer Science, University of Illinois at Urbana, 1304 West Springfield Dr., Urbana, Ill., 61801

Bruce W. Porter
Computer Science Department, University of Texas, Austin, TX 78712 USA

Alain Rappaport
Neuron Data, 444 High St., Palo Alto, CA 94301 USA

Edwina L. Rissland
Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003 USA

Valerie L. Shalin
Honeywell, Systems and Research Center, 3600 Technology Drive, Minneapolis, Minn. 55418 USA

Mildred L. G. Shaw
Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4

David B. Shema
Knowledge-Based Systems Laboratory, Advanced Technology Center, Boeing Computer Services, P.O. Box 24346, Seattle, WA 98124 USA

Kenneth Silvestro
Computer Science and Information Studies, Adams Hall, Bowdoin College, Brunswick, Maine 04011 USA

Padhraic Smyth
Department of Electrical Engineering 116-81, California Institute of Technology, Pasadena, CA 91125 USA

Jeffrey Stout
Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 USA

Deborah Tranowski
Advanced Decision Systems, 201 San Antonio Circle, Suite 286, Mountain View, CA 94040 USA

Craig C. Wier
Department of Speech Communication, University of Texas, Austin, TX 78712 USA

Edward J. Wisniewski
Honeywell, Systems and Research Center, 3600 Technology Drive, Minneapolis, Minn. 55418 USA

Ian H. Witten
Knowledge Sciences Institute, Department of Computer Science, The University of Calgary, 2500 University Drive NW, Calgary, Canada T2N 1N4

J. Brian Woodward
Knowledge Sciences Laboratory, Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4

Stefan Wrobel
Techn. Univ. Berlin, FR 5-8, Franklinstr. 28/29, 1000 Berlin 10, West Germany

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

Protos: An Exemplar-Based Learning Apprentice[1]

E. Ray Bareiss (bareiss@pheasant.cs.utexas.edu)
Bruce W. Porter (porter@fall.cs.utexas.edu)
Department of Computer Sciences, University of Texas, Austin, TX 78712 U.S.A.
Craig C. Wier (uthrl@ngp.utexas.edu)
Department of Speech Communication, University of Texas,
Austin, TX 78712 U.S.A.

Abstract

Building Protos, a learning apprentice system for heuristic classification, has forced us to scrutinize the usefulness of inductive learning and deductive problem solving. While these inference methods have been widely studied in machine learning, their seductive elegance in artificial domains (*e.g.* mathematics) does not carry-over to natural domains (*e.g.* medicine). This paper briefly describes our rationale in the Protos system for relegating inductive learning and deductive problem solving to minor roles in support of retaining, indexing, and matching exemplars. The problems that arise from "lazy generalization" are described along with their solutions in Protos. Finally, an example of Protos in the domain of clinical audiology is discussed.

1. Introduction

Learning and reasoning from exemplars is a promising alternative to inductive learning and deductive reasoning. Exemplar-based learning involves remembering and indexing specific training cases. Reasoning from learned exemplars involves interpreting a new case by recalling a similar exemplar for guidance. In contrast, inductive learning generalizes specific training cases to form an abstract description. Reasoning involves deducing that a new case is subsumed by the abstract description. While this distinction might appear inconsequential, it has far-reaching implications, ranging from psychological validity to the feasibility of machine learning programs for diagnostic expert systems.

Our development of Protos, a learning apprentice for heuristic classification tasks, provides impetus and context for the study of exemplar-based systems. The primary goal of our Protos research is to develop a general mechanism for acquiring classification knowledge while providing interactive assistance to users. The requirements for such a system forced us to reconsider the popular wisdom in machine learning that training should be compiled and compressed into generalizations. Before turning to our solution, we briefly review the major requirements of the Protos system.

[1] Support for this research was provided by the Army Research Office under grant number ARO DAAG29-84-K-0060 and the National Science Foundation under grant number IRI-8620052.

First, the polymorphy of natural concepts must be centrally addressed in both the knowledge representation and the algorithms for learning and problem solving. Artificial domains, which have been the primary focus for machine learning researchers, permit simpler representations and algorithms. Concepts in artificial domains can be "classically defined" [SMIT81] using a set of necessary and sufficient conditions. Examples of such concepts are "a forked position in chess" [MINT84], "a prime in number theory" [LENA76], and "a good problem-state for applying substitution in symbolic integration" [MITC83, PORT86B]. However, the significant variability among instances of natural concepts prevents inductive learning of a uniform set of necessary and sufficient conditions. For example, consider inductive learning of the concepts "science," "friend," or "pollutant." If general descriptions for such concepts *can* be formed, they will be so vague as to be useless for classification of new cases. Abstract definitions also preclude other uses of concepts such as exclusion of near-misses, generation of examples of concepts, explanation, guiding the interpretation of new cases, *etc.* [SCHA86, PORT86A]. Protos satisfies the requirement to address concept polymorphy by focusing on learning and indexing exemplars of each concept.

Second, classification of new cases has two interrelated requirements. The first requirement is that Protos must learn and apply concept models. The identity of new cases is frequently unclear due to concept polymorphy and incomplete or noisy case descriptions. Models can be effectively employed to guide the interpretation of individual cases [WEIS78, NII82]. Related to this is the requirement for efficient generation of explanations. An explanation relating a new case to a generalized concept is commonly constructed by deducing that the general concept entails the specific case [BUCH84, MITC86, DEJO86]. This can involve considerable deductive inference. The amount of inference required is a function of the "distance" between the new case, described in the instance language, and the concept description, described in the generalization language. These two requirements are satisfied in Protos by using learned exemplars, instead of general concepts, as the targets for matches with new cases. An exemplar can provide specific guidance in interpreting an unclear case and can be efficiently matched because of proximity.

The primary lesson of Protos is that learning-apprentice systems for heuristic-classification tasks should be "lazy generalizers." Inductive learning is "eager" generalization which assumes that (1) generalizations *can* be formed, (2) problem solving is primarily deductive inference from generalizations, and (3) classification is the primary application of learned knowledge. There have been numerous efforts to relax these assumptions, ranging from transformational-matching systems [KEDA85, MCCA81] to probabilistic systems [BUCH84, ZADE65]. We concluded that inductive learning and deductive classification should play only a minor role in Protos [PORT86A]. Exemplar-based learning and problem solving have considerable psychological support [BARE87, MEDI83A, SMIT81, BROO78] and together provide a unified solution to the problems faced by a learning apprentice system: learning

```
REPEAT
1  Input a case to be classified
2  Retrieve remindings based on the features of the case and heuristically
     combine them
   REPEAT
    REPEAT
3     Use the strongest combined reminding to select an exemplar from the
        category structure
4     Evaluate the similarity of the new case to the exemplar by
        constructing an explanation of their equivalence
      UNTIL an adequately explained match is found
5     Use difference links to improve the classification by considering
        neighboring exemplars as alternatives
6     Present the match and explanation to the teacher
      IF the teacher disapproves
7        THEN Request new domain knowledge from the teacher
8              Reassess the remindings which suggested the exemplar
   UNTIL the teacher approves
   IF the match is very strong
9     THEN Merge the case with the exemplar
10    ELSE Add the case to the category structure as a new exemplar
   IF the case was initially misclassified
11    THEN Install difference links to the exemplars involved

   UNTIL training is complete
```

Figure 1
The Protos Algorithm

and representing polymorphic concepts, classifying unclear cases, and efficiently generating explanations.

The next section describes the problems that are inherent in exemplar-based systems and discusses their solutions in Protos. We are in the process of applying Protos to the domain of clinical audiology. Section 3 details how Protos processed a typical case. Section 4 describes our preliminary evaluation of the performance of Protos as a learning apprentice.

2. Issues in Exemplar-based Systems and Their Solutions in Protos

Exemplar-based systems raise a number of questions. This section answers these questions in the context of the Protos algorithm shown in Figure 1. Examples are drawn from the Protos knowledge structure shown in Figure 2.

2.1 What Unites the Exemplars of a Category?

Category cohesiveness requires that the underlying commonality of the category members be explicit in the representation. In Protos, each exemplar of a category is represented by features and explanations of each feature's relevance to the category. For example, Figure 2 shows two exemplars of the category **chairs** and the explanations of their features. The network of domain knowledge with the embedded exemplars is called a *category structure* in Protos. Each node in the

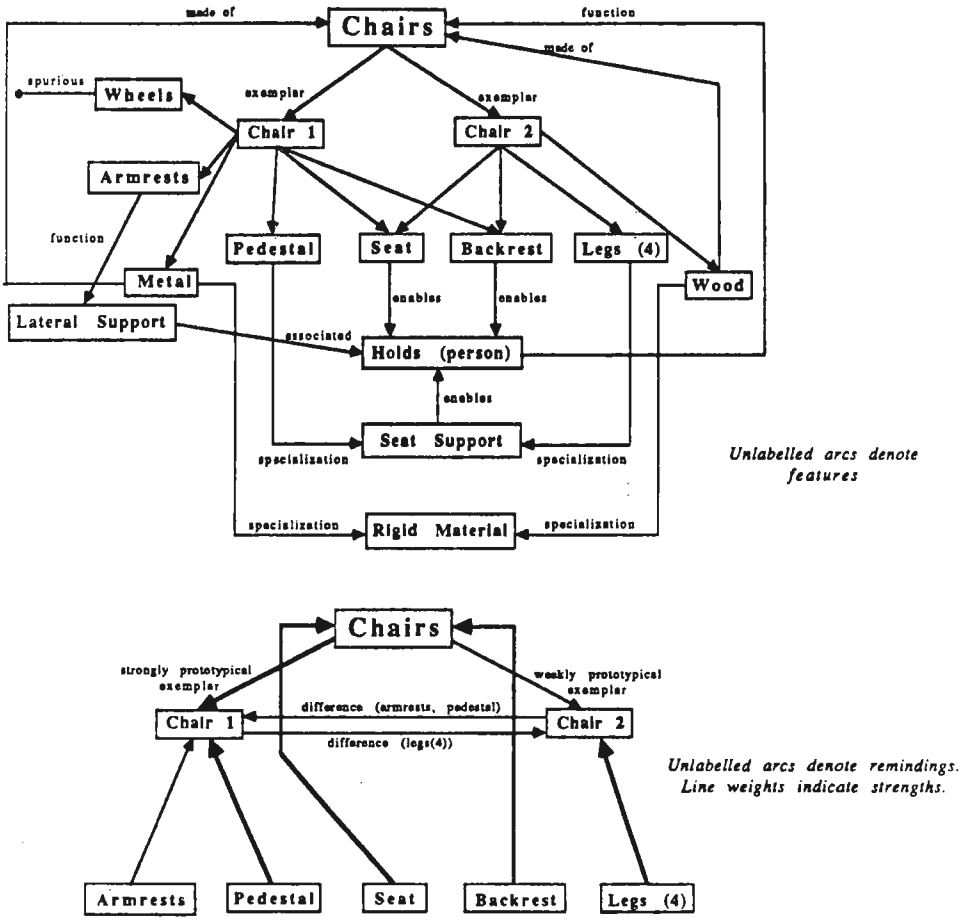


Figure 2
A Simple Category Structure
Top: Domain Knowledge Bottom: Indices for Exemplar Retrieval

category structure is either an exemplar-containing category or a domain primitive. For example, the node labelled *armrests* might contain exemplary armrests, each described with armrest features and explanations of their relevance.

2.2 How are the Exemplars Indexed for Efficient Retrieval?

The first step in processing a new case is to efficiently identify a matching exemplar. This section describes the three primary indexing mechanisms which Protos uses to retrieve exemplars. The methods for learning indices are described in section 2.5.

The first indexing mechanism relies on cues gleaned from features of the new case (see line 2 of the Protos algorithm). Each cue, called a *reminding* (*cf.*, [SCHA82, KOLO83]) is a link in the category structure from a feature to an exemplar or category (*cf.*, "cue validity of features" in [ROSC78, MEDI83B]). For example, the

feature *backrest* in Figure 2 provides a reminding to the category of **chairs** while the feature *pedestal* is idiosyncratic to the exemplar *chair 1*. The features of a new case typically provide numerous remindings which are heuristically combined into a few candidate classifications for the new case. For example, if the features of a new case include *backrest* and *pedestal*, Protos is reminded of the category **chairs** and the exemplar *chair 1*. These two remindings are heuristically combined, yielding a strong reminding to *chair 1*.

The second form of indexing relies on the prototypicality of exemplars with respect to their containing categories. Prototypicality provides a partial ordering on the exemplars of a category. The prototypicality of an exemplar is determined by *family resemblance*, the extent to which its features overlap those of other category members [Rosc75]. Protos may use prototypicality (see line 3 of the algorithm) to select an exemplar from a category in the set of candidate classifications. If there are remindings to a particular exemplar in the category, it is selected. Otherwise, an exemplar is selected from the category based on prototypicality. For example, given a new case which evokes remindings to **chairs** but no remindings to particular exemplars, Protos prefers *chair 1* based on prototypicality.

The third indexing mechanism is based on the differences between pairs of "neighboring" exemplars in the category structure (*cf.*, [Kolo83]). This mechanism is used in line 5 of the algorithm after Protos constructs a match with a retrieved exemplar (the subject of the next section) and notes featural mismatches. Protos attempts to improve the match by considering neighbors of the retrieved exemplar. Those *difference links* from the exemplar which are labelled with one or more of the noted mismatches are traversed to hill-climb to an improved match. For example, if a new case partially matches *chair 2* but has the additional feature of *armrests*, then the difference link to *chair 1* is traversed.

2.3 How are Exemplars Used During Problem Solving?

Each exemplar in the category structure is a model for interpretation of new cases. However, only those exemplars which are similar to the new case are useful. Initially, the remindings are used to retrieve an exemplar based on expectation of similarity. The true similarity is then determined by *knowledge-based pattern matching* (see line 4 of the algorithm). Knowledge-based pattern matching uses the domain knowledge in the category structure to construct an explanation of the equivalence of the exemplar and the new case. The explanation is the justification for concluding the presence of criterial features of the exemplar based on the observed features of the new case. For example, Protos might match the feature *legs*(4) in a new case to *pedestal* in the exemplar *chair 1* based on the knowledge:

$$legs(4) \overset{specialization}{\longrightarrow} seat\ support \overset{specialization}{\longleftarrow} pedestal$$

This suggests that the features are equivalent because each provides evidence for the functional feature, *seat support*.

Protos gauges the amount of effort to expend on knowledge-based pattern matching between a new case and a retrieved exemplar. Effort is determined by multiple factors, including the combined strength of the remindings from the features of the case to the exemplar, the prototypicality of the exemplar, and the degree of direct match between the case and the exemplar. In summary, knowledge-based pattern matching (deductive inference) is used in Protos to confirm expectations about a new case. The amount of effort expended in the process is restricted, lest Protos "see mirages."

2.4 How is the Category Structure Learned?

The category structure is learned from training provided by the teacher. The teacher provides cases for Protos to classify. When Protos fails, the teacher explains the correct answer (see line 7 of the algorithm). The cases described to Protos become exemplars and the accompanying explanations become domain knowledge in the category structure. Past research has ignored the role of explanations in training and assumed that a teacher could compensate for a narrow communication channel by either providing numerous training examples (*e.g.*, [QUIN86]) or relying on the learner's *a priori* knowlege (*e.g.*, [MITC86]).

Protos expects an explanation of the relevance of each feature of a new case to the category to which the case is assigned. If Protos is unable to generate an explanation for a feature then the teacher assists. The teacher provides domain knowledge in the form of terms and predefined relations, which Protos integrates into the category structure. For example, consider the case of a *hanging* chair suspended from the ceiling by ropes, with neither legs nor pedestal. Protos cannot form a close match with either *chair 1* or *chair 2* until the teacher provides the information in the form of an explanation that:

$$chair\ ropes \overset{specialization}{\longrightarrow} seat\ support.$$

2.5 How are the Indices for Exemplars Learned?

As described in section 2.2, there are three types of indices for exemplars in the category structure: remindings, prototypicality, and differences. Protos learns and adjusts these indices during the course of training. This section describes the learning process.

Remindings are heuristic estimates of the likelihood of particular exemplars or categories being relevant to the processing of a new case. Initially these remindings are learned analytically. A teacher-supplied explanation which strongly relates a feature to a category or an exemplar is compiled into a reminding link. For example, the explanation:

$$seat \overset{enables}{\longrightarrow} holds(person) \overset{function}{\longrightarrow} chairs$$

suggests that *seat* should remind Protos of the category **chairs**.

When a reminding associated with a feature does not contribute to finding a successful classification, Protos reassesses its validity. Since remindings are compiled by analyzing explanations, Protos attempts to reassess a reminding by checking the consistency of the reminding with both the current state of its domain knowledge and the competing remindings. If analysis does not indicate that the reminding should be adjusted, Protos weakens the reminding to reflect diminished confidence; a reminding which continues to be weakened over time is eventually removed.

Protos estimates prototypicality by noting an exemplar's successful use in problem solving. When an exemplar closely matches a new case, the exemplar exhibits family resemblance within its category. The more new cases which are successfully matched by the exemplar, the more prototypical it becomes.

If Protos misclassifies a case, it installs difference links in the category structure to avoid similar near misses in the future. As discussed above, difference links allow Protos to improve a tenative match by identifying neighbors of the matched exemplar which may provide better matches.

2.6 When are Exemplars Generalized?

As described in section 2.4, each new case presented by the teacher is integrated into the category structure. This integration may involve merging the new case with an existing exemplar. Protos attempts to merge the descriptions if knowledge-based pattern matching determines they are highly similar and the teacher concurs. Specifically, all of the important features of the exemplar must be accounted for by explanations to features of the new case. This means that Protos considers the exemplar and the new case to be equivalent with respect to its goals and training.

As a result of a merge, the features of the exemplar might be generalized. For example, consider the new case, *chair 2'*, which is the same as exemplar *chair 2* in Figure 2 except it is made of metal instead of wood. Protos would merge the new case with the exemplar after constructing the following explanation for the mismatched features:

$$metal \overset{specialization}{\longrightarrow} rigid\ material \overset{specialization}{\longleftarrow} wood.$$

This explanation suggests that the exemplar feature *wood* can be generalized to *rigid material* in the resulting merged exemplar.

Another important source of exemplar generality is knowledge-based pattern matching. An exemplar implicitly represents all of the cases (seen or unseen) which are reachable by transforming the exemplar's features using domain knowledge. Therefore, the generality of an exemplar can increase with the amount of domain knowledge in the category structure.

3. An Example of Classifying and Learning

This section demonstates how Protos learns while providing interactive assistance to the user/teacher. Our example is from the domain of clinical audiology which

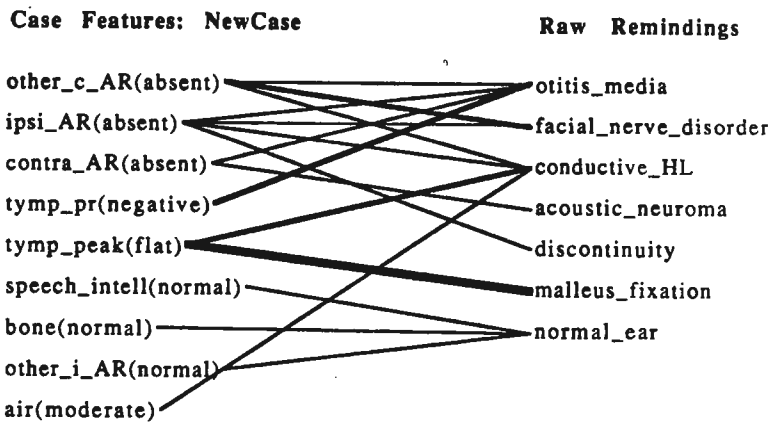


Figure 3
The Features of NewCase and
the Remindings to Diagnostic Categories

involves the evaluation and diagnosis of hearing disorders. We are currently training Protos with 200 sequential cases from Baylor College of Medicine. Each case presented to Protos consists of patient-reported symptoms, patient history information, and the results of routine tests. This is the data that a clinician considers when diagnosing a patient. We believe that clinical audiology is representative of a large set of heuristic classification tasks for which Protos is appropriate.

We now describe the twenty-fourth case of hearing disorder processed by Protos. For clarity, this case is called NewCase. Protos is presented with the features of NewCase which represent the results of tests of the patient's hearing and the biomechanical and physiological status of the patient's ear. These features are:

air(moderate)
bone(normal)
speech-intell(normal)
tymp-pr(negative)
tymp-peak(flat)
ipsi-AR(absent)
contra-AR(absent)
other-i-AR(absent)
other-c-AR(absent)

As shown in Figure 3, Protos is reminded of multiple diagnostic categories by the features of NewCase. Line weights indicate the strengths of the remindings. These "raw" remindings are heuristically combined to determine the most likely classifications for this case. As a result of this process, **otitis media** and **malleus fixation** emerge as the most likely categories.

The next step is to select an exemplar to serve as a model for NewCase. The reminding to **otitis media** is strongest, so Protos tries it first. Exemplar selection is based on prototypicality since Protos has no remindings to specific exemplars. Protos retrieves Case-17-1-R which is currently the most prototypical exemplar of **otitis media** in the category structure. The results of matching the two cases are illustrated in Figure 4(a). All of the featural matches are direct except for the

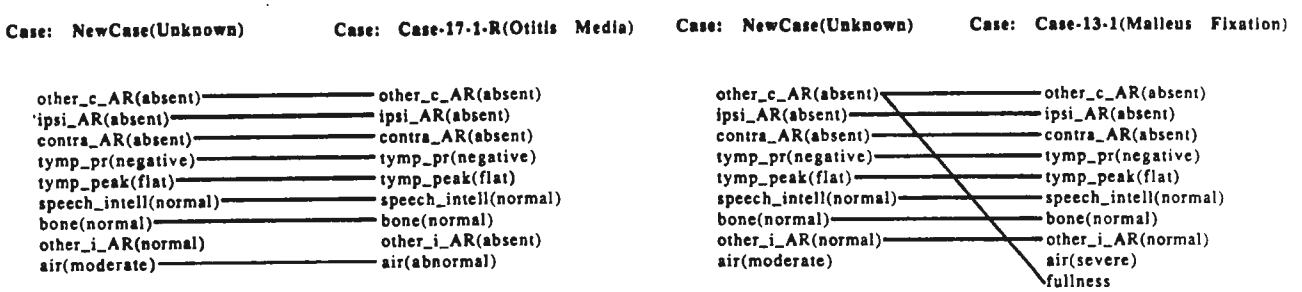


Figure 4
Featural Matches from Protos's Attempts to Classify NewCase
(a) The First Attempt (b) The Second Attempt

match between *air(moderate)* and *air(abnormal)*. This match is justified by the explanation *air(moderate) has generalization air(abnormal)*. Protos presents this evidence, but the teacher rejects the diagnosis. Protos asks the teacher if s/he agrees with the explanation of the match. The teacher approves of the reasoning but insists on considering the alternatives.

As a consequence of the teacher's rejection of the diagnosis, Protos weakens the remindings in the category structure which led to the misclassification. Protos then tries to match an exemplar of the category **malleus fixation**. Case-13-1 is selected as the most prototypical of this category. The result of using this case as a model for NewCase is illustrated in Figure 4(b). All featural matches are direct except for the match between *other-contra-AR(absent)* and *fullness*. This match is explained by:

other-c-AR(absent) co-occurs with conductive-HL which requires fullness.

Presented with all of the evidence, the teacher accepts the diagnosis of **malleus fixation**. In an effort to improve the overall match, Protos asks about the unmatched exemplar feature *air(severe)*. The teacher tells Protos that

air(moderate) has generalization air(abnormal) which has specialization air(severe).

Next Protos complains that the explanation linking *other-c-AR(absent)* to *fullness* is weak. Actually, the explanation is wrong, and the teacher corrects Protos by telling it that *conductive-HL only occasionally requires fullness.* The teacher then provides the correct explanation:

(tymp-peak(flat) AND air(moderate)) is usually sufficient for fullness.

Since the teacher has provided new information, Protos retries the match. Figure 5 shows the result, which is accepted by the teacher without further revision. Because all of the features of Case-13-1 are matched, Protos suggests that NewCase and Case-13-1 be merged. The teacher agrees. As a result of this decision, Protos's belief in the prototypicality of Case-13-1 is increased, and NewCase is not added to the category structure. During the merging operation, Protos notes the explanation:

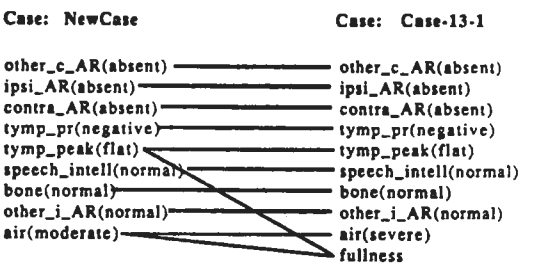


Figure 5
Revised Match Following the
Teacher's Correction of Protos' Explanation

air(moderate) has generalization air(abnormal) which has specialization air(severe)

and suggests that the feature *air(severe)* of Case-13-1 be generalized to *air(abnormal)*; the teacher concurs.

Since Protos found a mismatch to Case-17-1-R of **otitis media** before successfully matching Case-13-1 of **malleus fixation**, it suggests installing a difference link between the two exemplars to note how they differ. The teacher agrees and accepts this disposition of the case.

4. A Preliminary Evaluation of Protos

We are currently studying the effectiveness of the Protos approach for learning and applying heuristic classification knowledge. As described in section 3, our task domain is clinical audiology. One of the authors (Wier) is serving as the domain expert.

Professional training for clinical audiology typically requires two years of graduate school culminating in a Masters degree. Before becoming practicing clinicians, students must pass a national examination and serve a clinical-fellowship year following graduation. In the course of their graduate training, students see between 150-200 cases and then an additional 900-1200 cases in their clinical-fellowship year. To be consistent with this level of training, we decided that Protos should learn to perform demonstrably well given a training set of 200 cases. A database of 200 consecutive cases was obtained from the Audiology Clinic at Baylor College of Medicine in Houston, Texas. At this time, Protos has been presented the first 100 cases of its training set.

Protos has learned 16 diagnostic categories, which are represented with 86 exemplars. We were surprised that so many of the exemplars were retained in the category structure. Protos made little use of exemplar merging and generalization for two reasons. First, many of the categories are highly polymorphic and good generalizations (which are complete, consistent, and can be operationalized) do not exist. Second, a shortcoming in the explanation language prevented the domain

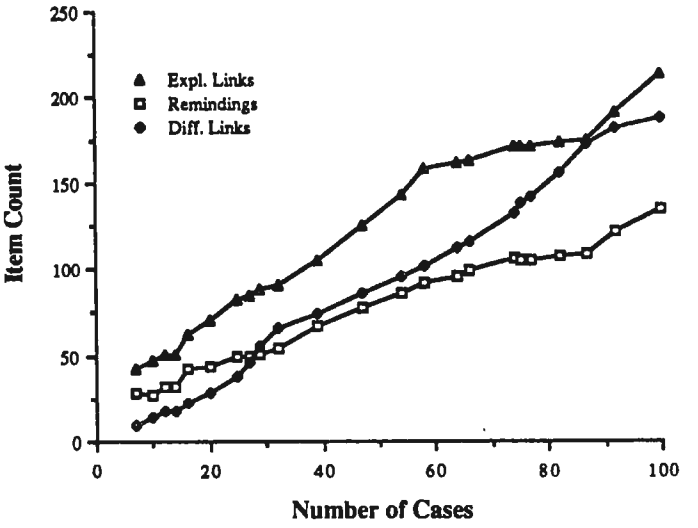


Figure 6
The Growth of the Protos Category Structure

Cases:		First Match	First or Second Match	Strongest Match	Strongest or Second Strongest
Training Cases	1-100	64%	90%	88%	94%
Training Cases	51-100	73%	93%	93%	100%
Test Cases	101-110	80%	100%	90%	100%

Figure 7
Protos' Classification Performance (Preliminary Results)

expert from giving *conditional* explanations of featural equivalence. In the earlier example, one of the teacher-provided explanations would be more correct were it expressed as:
IF the category is **malleus fixation** THEN
air(moderate) has generalization air(abnormal) which has specialization air(severe)
which equates the two values of *air* only in the context of this particular diagnosis. The expert estimates that this language facility would allow additional explanations enabling Protos to discard 30-40 of the retained exemplars. (Accordingly, a conditional explanation capability was incorporated into Protos and will be used in subsequent training.) The overall growth of the category structure as a function of the number of cases presented is shown in Figure 6.

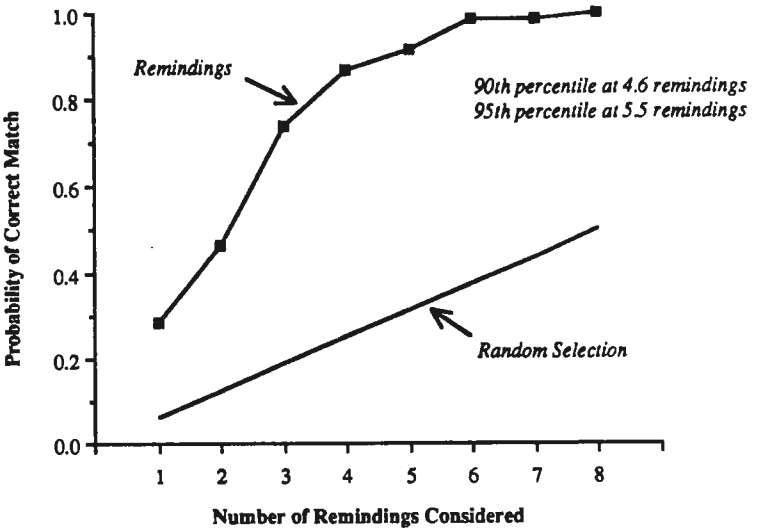


Figure 8
Probability of Correct Match as a Function of Remindings Considered

The primary measure of the performance of Protos is its accuracy in classifying new cases. Figure 7 summarizes this performance on the initial training.[2] After relatively little training, Protos' performance exceeds the performance that the expert estimates would be typical of a human student with similar experience. Throughout the learning process, Protos correctly classified about 90% of the cases. Much to our satisfaction, this performance was uneffected by continued growth of the category structure. After 100 training cases, Protos was presented with a normal mix of 10 test cases and demonstrated similar performance.

Another interesting measure of Protos' performance is the amount of effort expended to arrive at a correct classification. Figure 8 displays the probability of a correct match as a function of the number of remindings pursued by the system. For comparison, hypothetical performance as a result of randomly trying diagnostic categories is also plotted. We can estimate the amount of effort necessary to reach a classification by considering Figures 7 and 8 together. To be 90% correct in its classification, Protos has to consider only four or five of the 16 possible diagnostic hypotheses. However, not all of these hypotheses lead to finding matching exemplars. Most alternative hypotheses are quickly rejected because the case being classified does not have features considered to be important for category membership. Furthermore, in situations in which Protos finds two candidate matches, there is greater than 90% probability that the strongest match is the correct classification.

Protos is currently being taught with the second set of 100 cases of the hearing-disorder training set. More comprehensive analyses will be performed when the

[2] Since Protos is designed to perform heuristic classification (classifying cases into known categories) the first occurrence of each new category is excluded from this data.

process is complete. In addition, sensitivity analysis is planned to evaluate the effect of varying the initial indexing information provided by the remindings. Finally, applications to other domains are planned to demonstrate that Protos provides a general framework for learning to perform heuristic classification.

5. Summary

Protos is a learning apprentice for heuristic classification tasks. During the early design of Protos we concluded that inductive learning and deductive problem solving were unsuitable. The polymorphy of natural concepts coupled with the requirements for efficient classification and explanation precluded such simple solutions. We adopted an exemplar-based approach to learning and problem solving which was supported by considerable psychological evidence. We addressed the problems that arose from exemplar-based, "lazy generalization" and built the Protos system. While expert performance is not our primary concern, it is essential to our research paradigm to evaluate our methods by constructing and testing an expert system. This allows us to explore the "scaled-up" behavior of the methods. We are currently teaching Protos about clinical audiology and have found that it quickly evolves into an expert system with a modest training set.

Acknowledgements

The construction of Protos could not have succeeded without the generous assistance of Claudia Porter, Joe Ross and Todd Stock. Adam Farquhar and Ken Murray also provided useful comments. Professor James Jerger, of the Baylor College of Medicine in Houston, Texas, graciously allowed access to the audiology clinic records which provided the database used to evaluate Protos' performance.

References

[Bare87] Bareiss, E.R. and Porter, B.W. A Survey of Psychological Models of Concept Representation. Technical Report AI TR-87-50, Department of Computer Sciences, University of Texas at Austin.

[Broo78] Brooks, L. Non-Analytic Concept Formation and the Memory for Instances. Appearing in *Cognition and Categorization*, Rosch, E. and Lloyd, B.B. (ed.), Hillsdale, NJ: Erlbaum, 1978, pp. 169–211.

[Buch84] Buchanan, B.G. and Shortliffe, E.H. *Rule-Based Expert Systems (The MYCIN Experiments of the Stanford Heuristic Programming Project)*, Addison-Wesley, 1984.

[DeJo86] DeJong, G. and Mooney, R. Explanation-Based Learning: An Alternate View. *Machine Learning 1*, 2 (1986), 145–176.

[Keda85] Kedar-Cabelli, S. Purpose-Directed Analogy. *Proceedings of the Seventh Annual Conference of The Cognitive Science Society* (1985), 150–159.

[Kolo83] Kolodner, J. L. Maintaining Organization in a Dynamic Long-Term Memory. *Cognitive Science 7*, 4 (1983), 243–280.

[Lena76] Lenat, D.B. *AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search*, PhD dissertation, Computer Science Department, Stanford University, 1976.

[McCa81] McCarty, L.T. and Sridharan, N.S. The Representation of an Evolving System of Legal Concepts. *Proceedings of the International Joint Conference on Artificial Intelligence* (1981), 246–253.

[Medi83a] Medin, D.L., Dewey, G.I., and Murphy, T.D. Relationships between Item and Category Learning: Evidence that Abstraction is Not Automatic. *Journal of Experimental Pyschology: Learning, Memory and Cognition 9 (4)* (1983), 607–625.

[Medi83b] Medin, D.L. Structural Principles of Categorization. Appearing in *Interaction: Perception, Development, and Cognition*, Shepp, B. and Tighe, T.(eds.), Erlbaum, 1983, pp. 203–230.

[MINT84] Minton, S. Constraint-Based Generalization: Learning Game-Playing Plans from Single Examples. *Proceedings of the National Conference on Artificial Intelligence* (1984), 251–254.

[MITC83] Mitchell, T.M., Utgoff, P.E., Nudel, B. and Banerji, R. Learning by Experimentation: Acquiring and Refining Problem Solving Heuristics. Appearing in *Machine Learning*, Michalski, R.S., Carbonell, J.G., and Mitchell, T.M. (eds.), Tioga Publishing, 1983.

[MITC86] Mitchell, T.M., Keller, R.M., and Kedar-Cabelli, S.T. Explanation-Based Generalization: A Unifying View. *Machine Learning 1, 1* (1986).

[NII82] Nii, H.P., Feigenbaum, E.A., Anton, J.J. and Rockmore, A.J. Signal-to-Symbol Transformation: HASP/SIAP Case Study. *AI Magazine 3*, 2 (1982).

[PORT86A] Porter, B. and Bareiss, E.R. Protos: An Experiment in Knowledge Acquisition for Heuristic Classification Tasks. *Proceedings of the First International Meeting on Advances in Learning*, 28 July – 1 August, 1986, Les Arcs, France. Sponsored by Y. Kodratoff and R. Michalski, 159–174. (Also available as Technical Report AI-86-32 from the Computer Sciences Department, University of Texas at Austin.)

[PORT86B] Porter, B. and Kibler, D. Experimental Goal Regression: A Technique for Learning Problem Solving Heuristics. *Machine Learning 1*, 3 (1986).

[QUIN86] Quinlan, J.R. Induction of Decision Trees. *Machine Learning 1,1* (1986).

[ROSC75] Rosch, E. and Mervis, C.B. Family Resemblance Studies in the Internal Structure of Categories. *Cognitive Psychology 7* (1975), 573–605.

[ROSC78] Rosch, E. Principles of Categorization. Appearing in *Cognition and Categorization*, Rosch, E. and Lloyd, B.B. (ed.), Hillsdale, N.J.:Erlbaum, 1978.

[SCHA82] Schank, R.C. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*, Cambridge University Press, 1982.

[SCHA86] Schank, R.C., Collins, G.C., and Hunter, L.E. Transcending Inductive Category Formation in Learning. *The Behavioral and Brain Sciences* (1986) (to appear).

[SMIT81] Smith, E. and Medin, D. *Categories and Concepts*, Cambridge: Harvard University Press, 1981.

[WEIS78] Weiss, S.M., Casmir, A.K., and Amarel S. A Model-Based Method for Computer-Aided Medical Decision-Making. *Artificial Intelligence*, Volume 11 (1978), 145–172.

[ZADE65] Zadeh, L.A. Fuzzy Sets. *Information and Control 8*, 338–353.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

USES OF REPERTORY GRID-CENTERED KNOWLEDGE ACQUISITION TOOLS FOR KNOWLEDGE-BASED SYSTEMS

John H. Boose
Knowledge Systems Laboratory, Advanced Technology Center, Boeing Computer Services
P.O. Box 24346, Seattle, WA, 98124, USA

ABSTRACT

Repertory grid-centered knowledge acquisition tools are useful as knowledge engineering aids when building many kinds of complex knowledge-based systems.These systems help in rapid prototyping and knowledge base analysis, refinement, testing, and delivery.

These tools, however, are also being used as more general knowledge-based decision aids. Such features as the ability to very rapidly prototype knowledge bases for one-shot decisions and quickly combine and weigh various sources of knowledge make these tools valuable outside of the traditional knowledge engineering process.

This paper discusses the use of repertory grid-centered tools such as the Expertise Transfer System (ETS), Aquinas,Kitten, and KSS0. Dimensions of use are presented along with specific applications. Many of these dimensions are discussed within the context of ETS and Aquinas applications at Boeing.

1.0 REPERTORY GRID TOOLS

Some Repertory Grid-Based Tools

A growing number of knowledge acquisition tools incorporate repertory grid methods from Personal Construct Psychology (Kelly, 1955; Gaines and Shaw, 1981). Such tools include the Expertise Transfer System (ETS, Boose, 1984, 1985, 1986a), Planet (Gaines and Shaw, 1986), Aquinas (Boose and Bradshaw, 1987a, 1987b; Kitto and Boose, 1987), FMS Aid (Garg-Janardan and Salvendy, 1987), Kitten (Shaw and Gaines, 1987), Kriton (Diederich, Ruhmann, and May, 1987; Diederich, Linster, Ruhmann, and Uthmann, 1987), KSS0 (Gaines, 1987a, 1987b), and others. These tools represent problem-solving knowledge in repertory grids. Repertory grids hold ratings; each rating is a judgment by the expert relating a solution to a problem-solving trait. These tools interview experts directly and help them refine, expand, analyze, and test their problem-solving knowledge.

Aquinas and the Expertise Transfer System (ETS)

Aquinas, an expanded version of the Expertise Transfer System (ETS), is a knowledge acquisition workbench that supports several knowledge acquisition tasks. These tasks include eliciting distinctions, decomposing problems, combining uncertain information, incremental testing, integration of data types, automatic expansion and refinement of the knowledge base, use of multiple sources of knowledge, and providing process guidance. Aquinas interviews experts and helps them analyze, test, and refine their knowledge base. Expertise from multiple experts or other knowledge sources can be represented and used separately or combined. Results from user consultations are derived from information propagated through hierarchies. Aquinas delivers knowledge by creating knowledge bases for several different expert system shells. Help is given to the expert by a dialog manager that embodies knowledge acquisition heuristics.

Using Aquinas, small rapid prototypes of knowledge-based system can be built in as little as one hour, even when the expert has little understanding of knowledge-based systems or has no prior training in the use of the tool. Kelly's methods and theory provide a rich framework for modeling the qualitative and quantitative distinctions inherent in an expert's problem-solving knowledge.

Aquinas is discussed in more detail in Boose and Bradshaw (1987a). Aquinas is written in Interlisp and runs on the Xerox family of Lisp machines. Subsets of Aquinas also run in an Interlisp version on the DEC Vax and a micro computer "C/UNIX"-based portable version.

Repertory Grids and Knowledge Representation

Repertory grids are a basic unit of knowledge representation in these tools, although they are often combined with rules, semantic nets, logic, and so on.Experts enter and refine knowledge in the form of repertory grids, also referred to as rating grids. In a rating grid, problem solutions - *elements* - are elicited and placed across the grid as column labels, and traits of these solutions - *constructs* - are listed alongside the rows of the grid (Figure 1). Traits are first elicited by presenting groups of solutions and asking the expert to discriminate among them. Following this, the expert gives each solution a rating showing where it falls with respect to each trait, by default on a scale from 1 to 5. Then the expert assigns a relative importance to each trait.

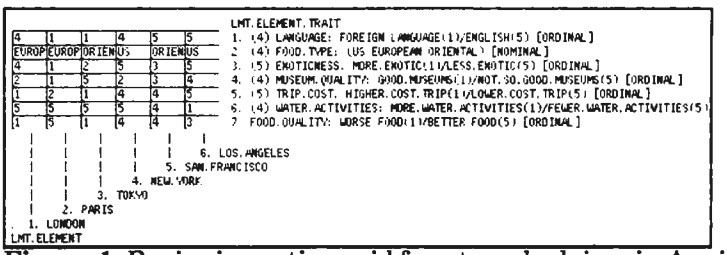


Figure 1. Beginning rating grid for a travel advisor in Aquinas.

After an initial grid is constructed, Aquinas, ETS, Kitten, and KSS0 help the expert refine and expand the knowledge base by invoking a variety of analysis tools. Similarities between elements and traits are analyzed to help the expert refine useful distinctions and eliminate those that are inconsequential or redundant. Inductive implications between trait values can show the expert higher levels of abstraction implied by a rating grid. If the expert disagrees with an implication, Aquinas helps the expert refine the grid. Distinctions captured in grids can be converted to other representations such as production rules, fuzzy sets, or networks of frames.

Extensions to Conventional Repertory Grid Methods

Conventional repertory grid methods have been significantly extended and combined with other techniques to help capture problem-solving knowledge more effectively.

Hierarchies. It can be difficult to represent certain types of problem-solving information in a single rating grid. In Aquinas, we extended repertory grid methods to help experts arrange rating grid knowledge in *hierarchies* and *lattices*. Tools in Aquinas help the expert build, edit, and analyze knowledge in hierarchies. These hierarchies allow the expert to break up complex problems into pieces of convenient size and similar levels of abstraction. Hierarchies in Aquinas are organized around *cases*, *knowledge sources* (i.e., experts), *solutions*, and *traits* (Figures 2 and 3).

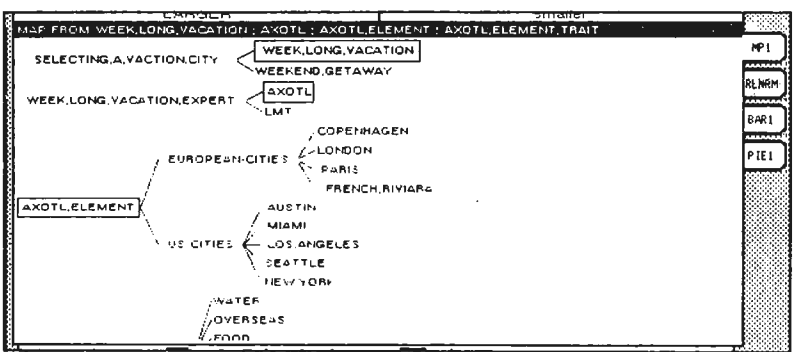


Figure 2. Portions of four hierarchies for a travel advisor in Aquinas. Two cases and two experts are shown, as well as expert Axotl's solution hierarchy for the WEEK.LONG.VACATION case.

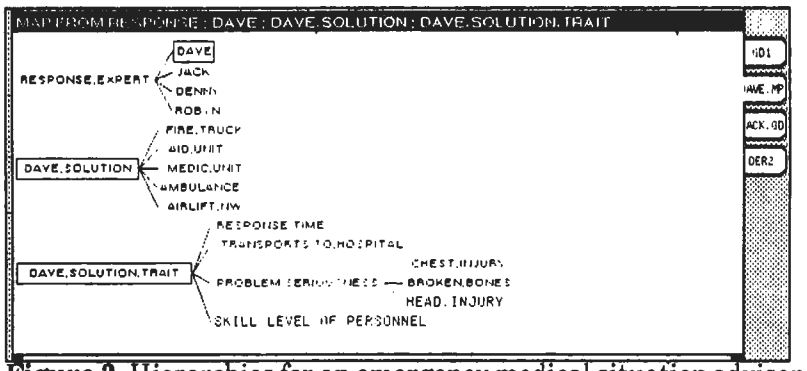


Figure 3. Hierarchies for an emergency medical situation advisor in Aquinas showing a portion of DAVE's knowledge.

Nodes in the four hierarchies combine to form rating grids. In the most simple case, the children of a node in a solution hierarchy supply the solutions along the top of a grid; the children of a node in a trait hierarchy supply the traits down the side of a grid. Rating values within the grid are judgments about the solutions with respect to each trait. These ratings may be entered directly or derived from judgments elsewhere in the hierarchies.

Strategies for helping the expert build and refine hierarchies in Aquinas include *laddering* (eliciting specializations and generalizations through tailored dialogs; Hinkle, 1965; Stewart and Stewart, 1981; Boose, 1986a), *cluster analysis* (experts are asked to label levels of abstraction represented by cluster junctions; see Figure 4), and *trait value examination* (certain combinations of trait values may suggest that hierarchical expansion is appropriate).

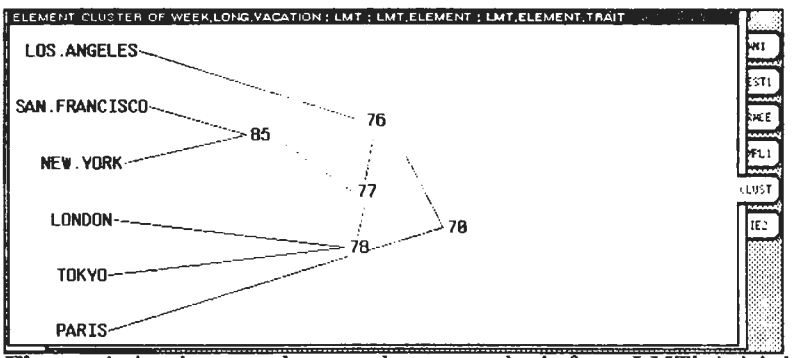


Figure 4. Aquinas produces a cluster analysis from LMT's initial travel advisor rating grid. LMT is asked to form hierarchies by labeling junctions on the cluster. One likely distinction might be made between domestic and foreign cities.

Multiple Trait Scales and Rating Distributions. In another extension to repertory grid methods, Aquinas allows non-ordinal traits to be represented in the rating grid. Nominal (unordered), interval, and ratio as well as ordinal traits can be used in a grid. Experts may assign arbitrary scales to ordered traits. Values of nominal traits appear in the grid, rather than numerical ratings (for example, "RED" may appear in a grid cell whose trait was "TEST-STAIN-COLOR" and whose legal values are "RED," "GREEN," and "BLUE"). Experts may also assign a distribution to values in a rating cell (for example, "RED .75 GREEN .25"). Aquinas' analysis tools and reasoning engine can handle distributions for all four trait scale types.

Rapid Prototypes of Consultation Systems. The knowledge embedded in hierarchical rating grids in Aquinas can be used to solve specific problems by running *consultations*. During a consultation, Aquinas asks the user to specify evidence and preferences with respect to traits for a given set of solution candidates. Consultation results are displayed as a ranked-list of solutions. Accompanying each solution is a set of numbers representing the strength of recommendation from knowledge sources, jointly and individually.

The model of problem-solving used in Aquinas is that of multiple knowledge sources (*experts*) that work together in a common problem solving context (*case*) by selecting the best alternatives for each of a sequential set of decisions (*solutions*). Alternatives at each step are selected by combining relevant information about preferences (relativistic reasoning), constraints (absolute reasoning) and evidence (probabilistic reasoning). In instances where the set of decisions are ordered with respect to solution refinement, this paradigm is similar to one suggested by Clancey who showed that many problems are solved by abstracting data, heuristically mapping higher level problem descriptions onto solution models, and then refining these models until specific solutions are found (Clancey, 1986). Methods for propagating evidence and preferences are described in (Bradshaw and Boose, 1987).

Experts and users can specify constraints in terms of absolute limits on the values of traits and interactions between these traits. For instance, the user can specify that a given solution must cost less than $400, or the expert can specify that the value of one trait depends on the value of others in certain circumstances.

Knowledge from multiple experts may be rapidly combined using Aquinas. Users may receive dissenting as well as consensus opinions from groups of experts, thus getting a full range of possible solutions. Disagreement between the consensus and the dissenting opinion can be measured to derive a degree of conflict for a particular consultation. Aquinas can be used for cost-effective group data gathering (Boose, 1986b).

Consultation results from the emergency medical situation advisor are shown below for a case where a fall resulted when a parafoil failed to open (Figure 5). The user has weighted each expert differently.Weighted solution contributions from each expert are shown. When only one expert contributes to a solution, the weight is shown at 1.00 (for example, AIRLIFT.NW). When more than one expert contributes to a solution each expert's contribution weight is shown. In MEDIC.UNIT for example DAVE, JACK, DENNY, and ROBYN each contribute to the solution with varying weights; a weighted average of their contributions is computed as the total weight for the solution.

```
Test results for consultation PARAFOIL-FALL

1 : AIRLIFT.NW ( 0.75 )        (DAVE 1.00 0.75)
2 : MEDIC.UNIT ( 0.72 )        (DAVE 0.50 0.75)(JACK 0.045 0.58)
                               (DENNY 0.25 0.73)(ROBYN 0.20 0.67)
3 : AID.UNIT ( 0.62 )          (DAVE 0.50 0.63)(JACK 0.045 0.56)
                               (DENNY 0.25 0.57)(ROBYN 0.20 0.67)
4 : AMBULANCE ( 0.43 ) (DAVE 0.50 0.31)(JACK 0.045 0.58) (DENNY
                               0.25 0.64)(ROBYN 0.20 0.42)
5 : FIRE.TRUCK ( 0.27 )        (DAVE 0.50 0.21)(JACK 0.045 0.48)
                               (DENNY 0.25 0.36)(ROBYN 0.20 0.25)
```

Figure 5. Consultations results in Aquinas show weighted contributions from multiple experts in an emergency medical aid advisor.

Next, a dissenting opinion is presented. Correlation scores for all experts are listed. A score of 1.0 would represent perfect ordering agreement (lists in the same order as the consensus opinion); a score of -1.0 would represent perfect disagreement (lists in opposite order). A score of 0.0 would mean that the lists are ordered randomly with respect to each other.

Results from the expert with the "worst" score (the lowest) are presented side-by-side with the consensus opinion appearing above (Figure 6). Only those solutions that the dissenting expert knows about are listed in the comparative consensus. The order of consensus is preserved.

```
Dissenting opinion for consultation  PARAFOIL-FALL

Correlation scores for all experts:
   ROBYN 0.80
   DENNY 0.80
   JACK 0.40
   DAVE 0.90

 JACK has the most dissenting opinion ( 0.40 correlation score)

      JACK              /  Consensus
  1 : AMBULANCE 0.58    /  MEDIC.UNIT 0.72
  2 : MEDIC.UNIT 0.58   /  AID.UNIT 0.62
  3 : AID.UNIT 0.56     /  AMBULANCE 0.43
  4 : FIRE.TRUCK 0.48   /  FIRE.TRUCK 0.27
```

Figure 6. Consensus correlation scores are shown for each expert, and then the expert whose solution list has the lowest correlation score is shown side-by-side with the consensus list.

Users can get an idea of the overall agreement or disagreement among experts by comparing their correlation scores. Users can also see the entire range of acceptable solutions by comparing the consensus and dissenting opinions. In this case, even the most dissenting expert agreed that sending a FIRE.TRUCK was a poor choice of aid, but there was disagreement about the best unit to send (it turned out that JACK came from a district where an ambulance was usually more available than a medic unit).

If there is agreement about a solution, the user gains more confidence that these are valid recommendations. If there are wide differences of opinion between experts, the user can employee other tools in Aquinas to compare the appropriate rating grids to see where the experts' opinions differed. For instance, differences in ratings in grids can be compared where there are common solutions and traits (based on the MINUS tool in Planet, Shaw and Gaines, 1986). In this example, three experts were in substantial agreement, while JACK disagreed. A *difference grid* showing the differences between two experts can be constructed (Figure 7).

```
Difference grid for JACK and DENNY

                          Traits:
 0.75 0.75 0.50 0     1: 0.50 ********** PROBLEM.SERIOUSNESS
 0.50 0.75 0.25 0.25  2: 0.44 ********* RESPONSE.TIME
 0.25 0    0    0     3: 0.06 * TRANSPORTS.TO.HOSPITAL
----------------------
 0.50 0.50 0.25 0.083  Total: 4.00 ( 0.33 average per cell)

Solutions:
1: 0.50 ********** AID.UNIT
2: 0.50 ********** AMBULANCE
3: 0.25 ***** FIRE.TRUCK
4: 0.08 ** MEDIC.UNIT
```

Figure 7. A *difference grid* in Aquinas shows the differences and similarities between two experts. Difference grids can be combined to produce difference and similarity totals for groups of experts.

The difference grid shows normalized rating differences and total differences between JACK's and DENNY's grids. They disagree most about the traits PROBLEM.SERIOUSNESS and RESPONSE.TIME, and about solutions AID.UNIT and AMBULANCE. It would be instructive to listen to them try to reach a consensus and fill in ratings on a smaller grid with just these two traits and solutions (Boose, 1986b).

Automatic Generation of Production Rules. Aquinas and ETS can transform rating grids into sets of production rules. These rules can be automatically reformatted for several expert system building tools (KEE, KS-300/EMYCIN, LOOPS, M.1, OPS5, S.1, and others; Boose, 1986a). Certainty factors for rules may be generated based on EMYCIN's certainty factor calculus (Adams, 1985). Certainty

factor values are based on the relative strength of the rating, the relative weight of the trait, and the overall size of the grids.

Process Guidance. A subsystem of Aquinas called the *dialog manager* contains pragmatic heuristics to guide the expert through knowledge acquisition using Aquinas. Its help is important in the use of Aquinas, given the complexity of the Aquinas environment and the many elicitation and analysis methods available to the expert. The dialog manager makes decisions about general classes of actions and then recommends one or more specific actions providing comments and explanation if desired. This knowledge is contained in rules within the dialog manager in Aquinas. A session history is recorded so that temporal reasoning and learning may be performed (Kitto and Boose, 1987a). The dialog manager helps novices learn to use Aquinas, resulting in more efficient prototyping. It can even help in identifying the problem class and suggest whether or not Aquinas would be useful in eliciting knowledge that would solve the problem (Kitto and Boose, 1987b).

2.0 USES OF REPERTORY GRID KNOWLEDGE ACQUISITION TOOLS AS KNOWLEDGE-BASED DECISION AIDS

Repertory grid-based tools are useful as knowledge engineering aids when building complex knowledge-based systems. These systems help in rapid prototyping and knowledge base analysis, refinement, testing, and delivery.

These tools, however, can also be used as more general *knowledge-based decision support systems*. Such features as the ability to very rapidly prototype knowledge bases for one-shot decisions and quickly combine and weigh knowledge from various sources make these tools valuable outside of the traditional knowledge engineering process.

Suitable Application Areas

Repertory grid-centered tools work best on *analysis* problems, or those portions of *synthesis* problems that can be reduced to analysis problems. Analysis problems are those whose solutions can be comfortably enumerated (classification, interpretation, diagnosis), while synthesis problems are those whose solutions are built up from components (configuration, design, planning). Even with this limitation, these tools can be applied to a wide range of application problems. Future versions of Aquinas will experiment with ways to acquire knowledge for synthesis problems (Boose and Bradshaw, 1987; see Discussion section below).

Dimensions of Repertory Grid-Centered Knowledge-Based Decision Making

The charts below show several dimensions of repertory grid-centered knowledge-based decision making aids, along with examples in each area. The dimensions of use shown in the charts are:

Source and user: The source of the knowledge may be the same or different than the user of the knowledge.
Source quantity: One person or more than one person may contribute to the knowledge base.
User quantity: One person or more than one person may use the knowledge base.
Knowledge base longevity: Knowledge bases may be used once and "thrown away" after some problem insight has been achieved or they may be used repeatedly, at different rates (continuously, hourly, daily, weekly, monthly, and so on).

Source of knowledge is not a User

		Source of knowledge: Single source	Source of knowledge: Multiple source
User of knowledge	Single user	*One source prepares knowledge for another user* Intelligence gathering Report preparation Feasibility analysis Primarily single use, some multiple-use KB's	*Group decision aid in support of a decision maker* Delphi-like information gathering Training preparation and evaluation Feasibility analysis Primarily single use, some multiple-use KB's
	Multiple user	*One source prepares knowledge for many other users* Traditional expert systems Questionnaire development Survey reporting Decision-report preparation Feasibility analysis Both single- and multiple-use KB's	*Many sources prepare knowledge for many other users* Commercial large knowledge access Timesharing / periodically updated & distributed knowledge bases Complex expert systems including multiple sources of knowledge Feasibility analysis Both single- and multiple-use KB's

Source of knowledge is a User:

		Source of knowledge: Single source	Source of knowledge: Multiple source
User of knowledge	Single user	*One source prepares knowledge for own decisions* Personal decision aid: business, military, personal use One-shot decision help Feasibility analysis Both single- and multiple-use KB's	*Group of sources prepares knowledge for one of its members* Marketing, data collection where expert is one of the users Feasibility analysis Both single- and multiple-use KB's
	Multiple user	*Single source prepares knowledge for self and others* Close working group decision aid Rapid advice gathering Sales support group Feasibility analysis Both single- and multiple-use KB's	*Community of sources prepares knowledge for its own use* Participant systems "Living" (dynamic) knowledge bases for community use Exchange grids (counseling, point of view exchange, negotiation arbitration) Both single- and multiple-use KB's

Other dimensions of use of repertory grid-centered knowledge-based decision aids are:

Knowledge base update frequency: Knowledge may be updated with varying rates of frequency. For example, a "frozen" knowledge base may be used for a long time before it is updated or a "living" knowledge base may be updated continuously (a large, commercial, time-share knowledge base with few updates; a marketing trends knowledge base with periodic updates; a stock market knowledge base with continuous updates).

Stand-alone: The tool can be used for stand-alone decision use or used with other software systems (expert system shell; spreadsheet; database management system).

Commercial knowledge base use: The knowledge base can be generated for commercial, external or internal, or personal use.

Consulting Use: The decision maker can use the tool directly or a consultant can use the tool in support of the decision maker (decision consulting; education and training).

Empty tool or primed tool: The tool can be used as an empty general decision aid or primed with knowledge for specific applications. Knowledge templates or full or almost-full knowledge bases may be delivered to a customer. Some examples are knowledge for specific domains at varying levels of abstraction (general diagnosis, engine diagnosis, jet engine diagnosis), proprietary or commercially valuable knowledge, surveys (fill in the ratings and return the grid to the surveyors), or use of domain-tailored dialogs (such as the original use of repertory grids in psychotherapy where patients filled in specific roles based on supplied descriptions).

Process and product: These tools can be used in the process of making a knowledge base as well as actually delivering knowledge (feasibility analysis, education and training of experts in the knowledge engineering process).

Following are example tool use alternatives employing different combinations of the above dimensions.

1. **Decision aid for for one-shot decisions**
   A grid tool is used to help people gain insight while making decisions: What employee should I assign to this project? What stocks should I invest in? What car should I buy? How can I better represent my products to my customers?
2. **Stand-alone knowledge-based system development and delivery tool**
   A grid tool is used to help develop, deliver, and maintain a knowledge-based system in a cost-efficient manner.
3. **Group decision aid**
   A grid tool is used as an aid to facilitate rapid, documented group decisions: On what software and hardware environment should this project be implemented? What research should our company pursue? How should capital assets be assigned?
4. **Large-scale on-line knowledge bases for a community of users**
   A grid tool can be used to develop and maintain large dial-in knowledge-and-data bases for a large number of users (stock market advising, insurance claims information).
5. **Feasibility analysis and project exploration tool**
   A grid tool is used to help assess project feasibility: What project idea is technically feasible? What project ideas are possible in this general area? Given a project idea, what kinds of knowledge will be involved? Does it make sense to invest resources in this project?.
6. **Expert system building tool (shell) front-end**
   A grid tool can be used as a front-end to develop knowledge for another expert system shell (S.1, M.1, OPS5, etc.).
7. **Teaching aid**
   A grid tool can be used to help teach others (especially experts and beginning knowledge engineers) quickly and painlessly about the concepts involved in knowledge-based systems and knowledge engineering: What are production rules? What is a consultation system? How can an expert's knowledge be structured and modified? How can systems be tested and validated?
8. **Situation Insight**
   A grid tool can be used to discover important or controlling factors in a situation; these factors can help the user understand, control, or change the situation (job satisfaction and motivation self-analysis, psychotherapy, counseling, relations with colleagues).

Past Applications of ETS and AQUINAS

ETS (based on single grids) and Aquinas (handling multiple hierarchical grids) have been used to generate hundreds of small and medium sized knowledge-based systems. Some of these systems are listed below:

Aerodynamic Analysis Software - Geometry Advisor
Aerodynamic Analysis Software - Back End Analyzer
AI Book Advisor
AI Library Advisor
AI Source Consultant
AI Vendor Consultant
Aircraft Fault Isolator
Airplane Design Flutter Analyzer
Alphanumeric Character Recognizer (by feature)
Artificial Intelligence Citation Tutor
Automated Numerical Control - Cutter Consultant
ATLAS Structural Analysis Advisor
B1 Diagnostic Consultant
Bicycle Selector
Business Computing Needs Advisor
Camera Selector
Character Recognizer
Civic Project Advisor
Climbing - Practice Locations Selector
Color Consultant
Community Volunteer Activities Advisor
Computer Communications System Bug Advisor
Conference/Workshop Location Selector
Contract Award Advisor
Crime Avoidance Advisor
Database Management System Consultant
Documentation Update Consultant
Election Outcome Consultant
Employee Rank Orderer
Energy Aircraft Advisor
Expert System Application Advisor
Failure Modes and Effects Analyzer
Finish and Corrosion Control Consultant
Flight Mode Manager
Geology Analyzer
Golf Swing Correction Consultant
GT-STRUDL Structural Analysis Advisor
Hot-line Consulting Aid
Helicopter Avionics Diagnostic Aid
Helicopter Stick Position Advisor
House Plant Disease Identifier
Ice Cream Flavor Selector
Intelligence Gathering and Reporting
Japanese Restaurant Advisor (Seattle)
Jet Engine Fault Isolator
Lunch Nutrition Advisor
Materials Technology Advisor / Database Pilot
Merit Raise Advisor
Micro Computer - System Recommendations Consultant
Micro Computer Trouble Shooter
Motorcycle Selector
Mystery Novel Selector
Negotiations Advisor - Unreasonable Offer Responses
Negotiations Advisor - Joint Gain
Negotiation Arbitration Environment
Office Assignment Aid
Office Automation Workstation Advisor
Organization Development Intervention Consultant
Parts Quality Control Consultant
Personality Disorder Advisor
Personnel Resource Management Decision Aid
Palo Alto Restaurant Advisor
Plant Closure Advisor
Portfolio Management Aid
Product - Comparative Analysis
Program Bug Advisor (FORTRAN)
Programming Language Applications Advisor
Propulsion System Advisor
Publication Advisor
Radio Diagnostician
Relational Database Construction Advisor
Research Project Priority Advisor
Reporting Assignment Advisor
Robotic System Application Advisor
Sales Support Information Analysis
Silhouette Recognizer
Software Management Consultant
Software Release Advisor
Space Station Window Materials Consultant
Aerodynamic Analysis Software - Front End Advisor
Age Guesser (by facial features)
AI Course Advisor
AI Market Analysis Advisor
AI Tools Advisor
Aircraft Identifier
Airplane Mass Property Estimation Risk Analyzer
Airplane Noise Certification Advisor
Animal Identification Advisor
Associates' Course Advisor
Automatic Flight Controls Diagnostic Aid
AXOTL Software Advisor
Bank Examiner
Bond Durability Consultant
Business Graphics Package Consultant
Car Engine Diagnostic Aid
Car Horn Diagnostic Aid
Climbing - Gear Advisor
Cloud Identifier
Cookie Selection Consultant
Composite Materials Advisors
Computer Languages Consultant
Conflict of Interest Advisor
Counseling Aid
Curriculum Advisor
Delphi Group Information Gathering
Eigenvalue Solution Advisor
Employee Evaluation
Energy Control System Model Evaluator
Exercise Consultant
Expert System Tools Advisor
Finish Advisor for Design Engineers
Flight Controls Human Factors Assistant
Furniture Workstation System Consultant
Gift Advisor
Graphics Package Advisor
House Selector
Halloween Costume Consultant
Helicopter Hover Advisor
Helicopter Vibration Diagnostic Aid
House Plant Selector
Image Feature Analysis
Investment Advisor
Jet Engine Diagnostic Aid
Jet Engine Manual Advisor
Management Motivation Analyzer
Meal Advisor
Micro Computer - Needs Analysis
Micro Computer - Workstation Configuration Advisor
Molded Rubber Seal Advisor
MugWump Political Advisor
Navigation System Advisor
Negotiations Advisor - Information Seeking
Negotiations Advisor - Obtaining Leverage
Newspaper Selection Consultant
Office Automation System Advisor
Offshore Shop Activity Analyzer
Organizational Climate Diagnostician
Personal Computer Advisor
Personality Factor Advisor
Personnel Selection Consultant
Planning Paradigm Classifier
Plant Location Selector
Product Design and Impact Advisor
Product Marketing Advisor
Programming Language Advisor
Programming Language Evaluator and Selector
Psychiatric Diagnostic Consultant
Questionnaire Development
Rapid Software Prototyping Aid
Research and Development Lab Site Advisor
Resin Advisor for Composite Parts
Rivet Selector
Robotic Tool Selector
Seattle Restaurant Advisor
Software Bug Diagnosis and Prescription Consultant
Software Quality Advisor
Software Services Advisor
Statistics Package Use Advisor

Structural Analysis Software Selector
Sushi Selector
Task Priority Manager
Technical Sales and Services Consultant
Toronto Restaurant Advisor
Training Curriculum Advisor
Travel Agent for Domestic Cities
Typewriter Disagnostic Advisor
Vehicle Selector
Visual Target Identifier
Wine Advisor
Word Processing System Consultant

Survey Reporting and Analysis
Task Assignment Consultant
Tax Regulation Consultant
Telephone Feature Selection Consultant
Training Course Evaluator
Transport Airplane Configuration Selection Advisor
Travel Agent for Foreign Cities
Vacation City Advisor
Velocity Analysis Advisor
Weekend Visitor Activities Consultant
Wood Identifier

Structuring Hierarchies - Examples from the Analytic Hierarchy Process

Saaty's Analytic Hierarchy Process (AHP) is a scheme for organizing relations between problem elements and measuring their mutual influence through the use of pairwise comparisons (1980). A modified form of the measuring algorithm is incorporated in Aquinas' reasoning engine, and the structuring scheme can be directly represented in Aquinas' hierarchies.

Saaty has used the Analytic Hierarchy Process in a variety of practical situations, including prediction, resource allocation, planning, conflict resolution, and others. Many of the examples presented in his book show the use of AHP as a one-shot governmental decision aid. Some examples he presents are:

Measuring the world influence of nations
Setting priorities for different industries within a government
Setting priorities for transportation projects
Energy allocation among different sectors
Predicting average sizes of families
Optimum land use
Sector outputs vs. domestic needs
Resource allocation - selecting a method of transport across a river based on a cost-benefits analysis (tunnel vs. bridge vs. ferry)
Allocation of fuel under shortage constraints
Resource development priorities for a developing nation
Planning - comparing scenarios involving higher education in the US (1985 - 2000)
Planning - developing a transportation network in the Sudan
Electric power utility strategic planning
Optimum choice of coal plant technology
Energy storage system selection
Estimating the annual KWH consumption of home appliances

Estimating impact of inflation, recession, and energy cost on future sales
Project implementation vs. project deferral
Corporate planning - projected futures and desired futures
Conflicts in health care management
Beverage container material selection (material type, recycle or not)
Priorities of factors in changing the steel industry

Strategizing about the long term health of a college in terms its of curriculum
Selecting a school to attend
Promotion and tenure decisions
Priorities of influences on child behavior
Estimating distances
Psychotherapy and self-analysis
Job selection

This chart shows typical hierarchical levels used by Saaty for organizing information for different general application areas (conflict, planning, investment, and so on). In Aquinas, the first five columns are best represented as trait hierarchies, and the last two columns as solution hierarchies.

Generic for a system	Environmental constraints	Perspective (actors)	Objectives of actors	Policies	Outcomes	Resultant outcomes
Hierarchy for conflict	Constraints	Actors	Objective	Policies	Outcomes	Compromise or stable outcome
Forward or projected planning	Present --> organizational policies	Other --> actors	Other actor -> objectives	Policies -->	Scenarios -->	Logical future
Backward or idealized planning	Organizational response policies <-	Other <-- actors	Other actor <- objectives	Other actor <-- policies	Scenarios <--	Desired future
Portfolio cost-benefit analysis	Criteria	Sub-criteria	Objectives	Policies	Options	Best option or mix
Investment choice	Risk level	Major forces	Criteria	Problem areas projects	Specific	
Prediction	Risk level	Major forces	Criteria	Problem areas	Categories	

Saaty makes certain assumptions about the applicability of AHP to problem domains. These assumptions would also be valid when using Aquinas to structure problems in a similar way:

1. A system should be decomposable into comparability classes (the components) in the framework of a directed network.
2. Elements in each component may be compared with respect to some or all the elements in an adjacent component.
3. Comparisons can be made in terms of an absolute numerical scale to form ratios.
4. The priority or composite index of dominance of an element can be derived through a composition or weighting principle.
5. Any element in the hierarchy is considered relevant, although its priority may be low.

Example Business Applications

Stewart and Stewart (1981) list other business applications that have successfully taken advantage of repertory grid methods:

Market research - product design, impact of a new product, graduate recruitment, product advertising impact
Quality control - training new operators, resolving differences between operators
Questionnaire design - career choices, wastage studies, employee retention, performance appraisal, meeting attitudes, training effectiveness, office morale, consumer surveys
Motivation at work - employees examine their motivation by comparing attitudes toward different workplace events
Organizational climate - organizational culture, managerial effectiveness
Training evaluation - training needs, trainees' changes during and after training, trainees' and trainer's views of the success of training events, managers' views of training courses, management's view of training as a function
Counseling - problems in professional interpersonal relations, management development, coping with stress, career interviewing, manager-employee relations
Counseling using shared grids - group trait elicitation, marriage counseling, team building (with customers as solutions, for team building in a sales team; with clients as solutions, for a consultancy team; with school children as solutions, for teaching staffs; with trainees, for training staffs; with cases as solutions, for social worker teams; with products as solutions, for a design team; with faults as solutions, for inspection teams; with projects as solutions, for service staff teams)

Tailoring Grid Tools for Domain-Specific Use

Aquinas can be prepared with domain knowledge templates. These can take the form of reworded prompts, suggestion lists and templates for solutions, traits, and hierarchies, or partially complete knowledge bases. The Planet system allows separate files for building tailored dialogs.

Some domain-specific uses require modifications to Aquinas. A small database interface and a process control interface will be implemented and reported on in the near future.

Example Grids

Following are example rating grids from several Aquinas applications.

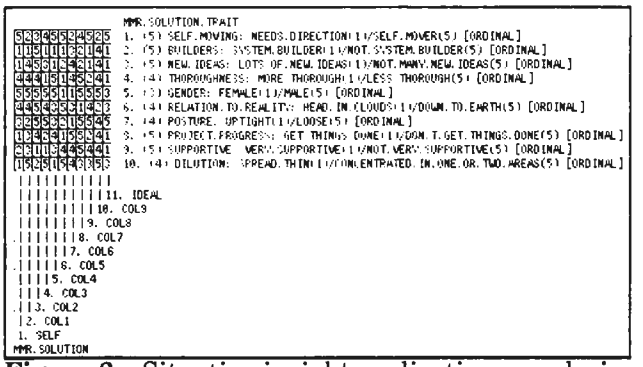


Figure 8a. Situation insight application - exploring relationships with colleagues: rating grid.

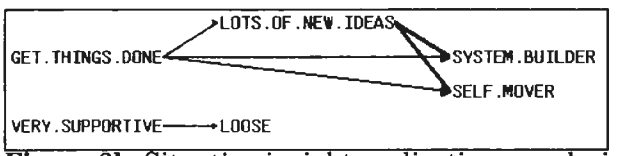


Figure 8b. Situation insight application - exploring relationships with colleagues: implication analysis.

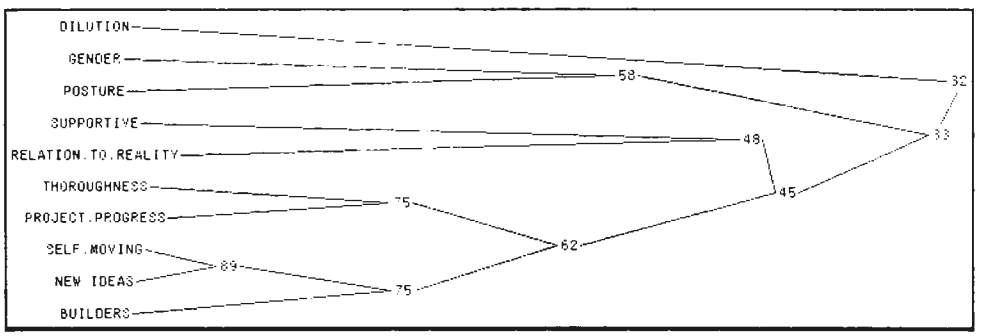


Figure 8c. Situation insight application - exploring relationships with colleagues: trait cluster.

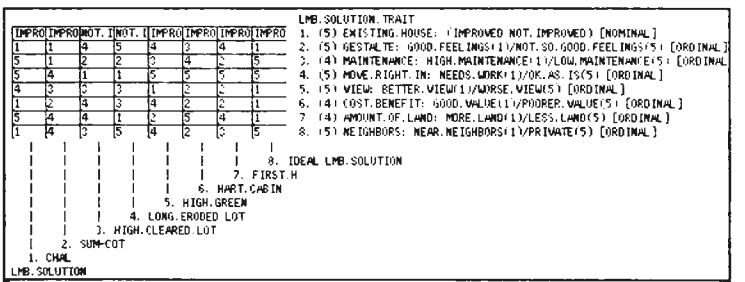


Figure 9. Personal decision aid - buying a home: rating grid.

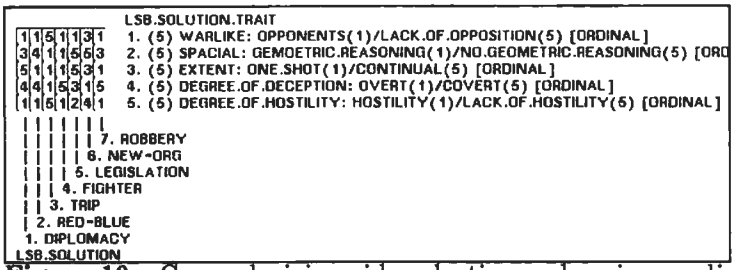


Figure 10a. Group decision aid - selecting a planning application: rating grid of expert LMB.

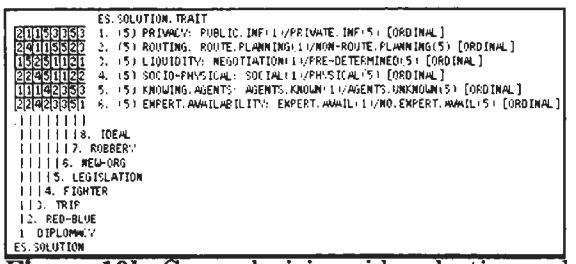


Figure 10b. Group decision aid - selecting a planning application: rating grid of expert ES.

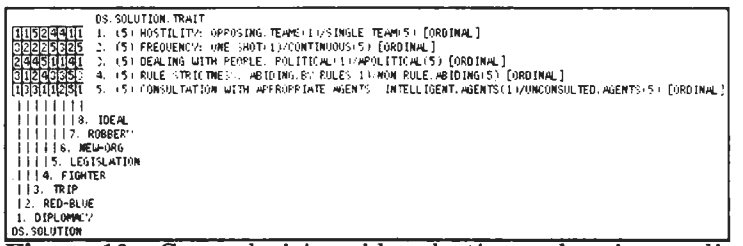


Figure 10c. Group decision aid - selecting a planning application: rating grid of expert DS.

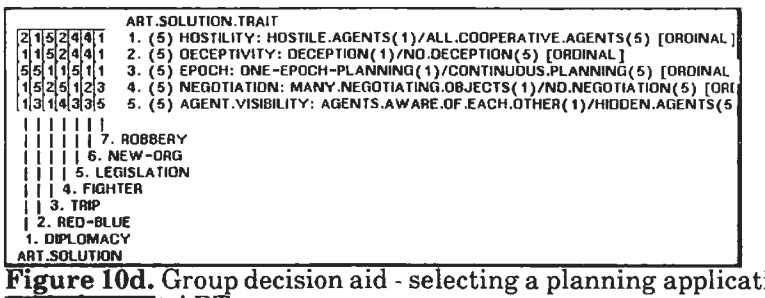
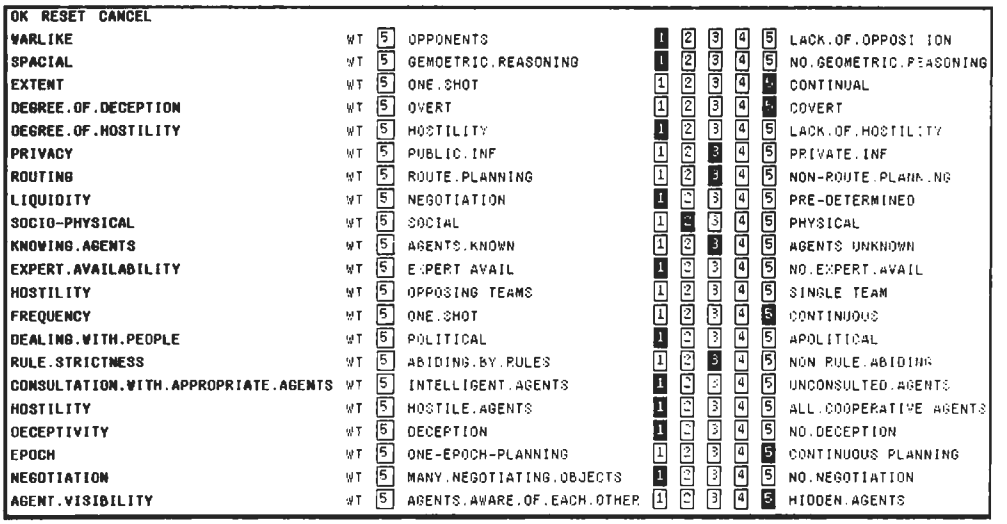


Figure 10d. Group decision aid - selecting a planning application: rating grid of expert ART.



OK RESET CANCEL										
WARLIKE	WT	5	OPPONENTS	1	2	3	4	5	LACK.OF.OPPOSITION
SPACIAL	WT	5	GEMOETRIC.REASONING	1	2	3	4	5	NO.GEOMETRIC.REASONING
EXTENT	WT	5	ONE.SHOT	1	2	3	4	5	CONTINUAL
DEGREE.OF.DECEPTION	WT	5	OVERT	1	2	3	4	5	COVERT
DEGREE.OF.HOSTILITY	WT	5	HOSTILITY	1	2	3	4	5	LACK.OF.HOSTILITY
PRIVACY	WT	5	PUBLIC.INF	1	2	3	4	5	PRIVATE.INF
ROUTING	WT	5	ROUTE.PLANNING	1	2	3	4	5	NON-ROUTE.PLANNING
LIQUIDITY	WT	5	NEGOTIATION	1	2	3	4	5	PRE-DETERMINED
SOCIO-PHYSICAL	WT	5	SOCIAL	1	2	3	4	5	PHYSICAL
KNOWING.AGENTS	WT	5	AGENTS.KNOWN	1	2	3	4	5	AGENTS UNKNOWN
EXPERT.AVAILABILITY	WT	5	EXPERT AVAIL	1	2	3	4	5	NO.EXPERT.AVAIL
HOSTILITY	WT	5	OPPOSING TEAMS	1	2	3	4	5	SINGLE TEAM
FREQUENCY	WT	5	ONE.SHOT	1	2	3	4	5	CONTINUOUS
DEALING.WITH.PEOPLE	WT	5	POLITICAL	1	2	3	4	5	APOLITICAL
RULE.STRICTNESS	WT	5	ABIDING.BY.RULES	1	2	3	4	5	NON RULE.ABIDING
CONSULTATION.WITH.APPROPRIATE.AGENTS	WT	5	INTELLIGENT.AGENTS	1	2	3	4	5	UNCONSULTED.AGENTS
HOSTILITY	WT	5	HOSTILE.AGENTS	1	2	3	4	5	ALL.COOPERATIVE AGENTS
DECEPTIVITY	WT	5	DECEPTION	1	2	3	4	5	NO.DECEPTION
EPOCH	WT	5	ONE-EPOCH-PLANNING	1	2	3	4	5	CONTINUOUS PLANNING
NEGOTIATION	WT	5	MANY.NEGOTIATING.OBJECTS	1	2	3	4	5	NO.NEGOTIATION
AGENT.VISIBILITY	WT	5	AGENTS.AWARE.OF.EACH.OTHER	1	2	3	4	5	HIDDEN.AGENTS

Figure 10e. Group decision aid - selecting a planning application: consultation trait preferences for an ideal application. This table was built from traits combined together from all four experts.

```
Test results for consultation  TEST-MULTI-PLANNING

1 : IDEAL ( 1.00 )              (ES 0.50 1.00)(DS 0.50 1.00)
2 : DIPLOMACY ( 0.81 ) (LSB 0.25 0.85)(ES 0.25 0.79) (DS 0.25
                                0.85)(ART 0.25 0.75)
3 : LEGISLATION ( 0.71 )        (LSB 0.25 0.65)(ES 0.25 0.75)
                                (DS 0.25 0.85)(ART 0.25 0.60)
4 : RED-BLUE ( 0.60 )           (LSB 0.25 0.60)(ES 0.25 0.58)
                                (DS 0.25 0.50)(ART 0.25 0.70)
5 : ROBBERY ( 0.60 )            (LSB 0.25 0.70)(ES 0.25 0.58)
                                (DS 0.25 0.40)(ART 0.25 0.70)
6 : FIGHTER ( 0.56 )            (LSB 0.25 0.80)(ES 0.25 0.46)
                                (DS 0.25 0.55)(ART 0.25 0.45)
7 : NEW-ORG ( 0.52 )            (LSB 0.25 0.25)(ES 0.25 0.79)
                                (DS 0.25 0.70)(ART 0.25 0.35)
8 : TRIP ( 0.30 )               (LSB 0.25 0.20)(ES 0.25 0.50)
                                (DS 0.25 0.35)(ART 0.25 0.15)

Dissenting opinion for consultation  TEST-MULTI-PLANNING

Correlation scores for all experts:
   ART 0.89
   LSB 0.68
   DS 0.79
   ES 0.60

ES has the most dissenting opinion
       ( 0.60 correlation score)

  ES                  /  Consensus
  1 : IDEAL 1.00      /  IDEAL 1.00
  2 : NEW-ORG 0.79    /  DIPLOMACY 0.81
  3 : DIPLOMACY 0.79  /  LEGISLATION 0.71
  4 : LEGISLATION 0.75 /  RED-BLUE 0.60
  5 : RED-BLUE 0.58   /  ROBBERY 0.60
  6 : ROBBERY 0.58    /  FIGHTER 0.56
  7 : TRIP 0.50       /  NEW-ORG 0.52
  8 : FIGHTER 0.46    /  TRIP 0.30
```

Figure 10f. Group decision aid - selecting a planning application: consultation results with consensus and dissenting opinions.

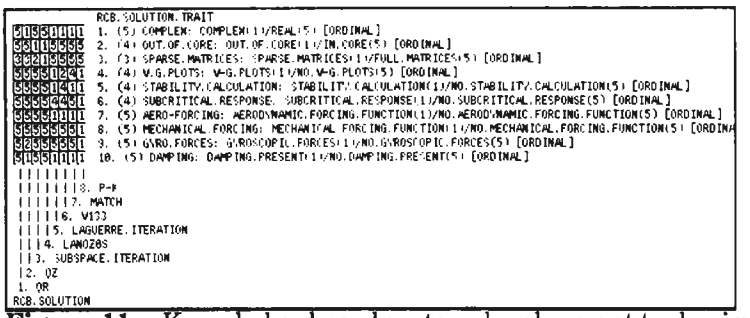


Figure 11a. Knowledge-based system development tool - eigen solution methods selector: rating grid.

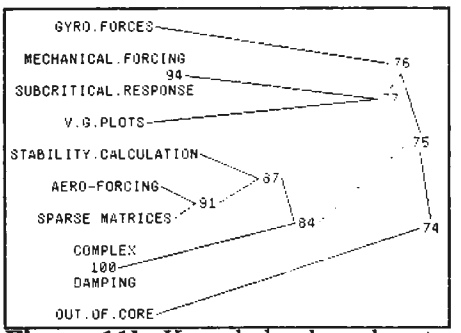


Figure 11b. Knowledge-based system development tool - eigen solution methods selector: trait cluster analysis.

3.0 DISCUSSION

Repertory grid-based tools are useful as decision aids in a variety of contexts - not just the conventional role of a knowledge acquisition tool helping build a complex expert system. In the last thirty years, repertory grid-based tools have been used for hundreds of different applications, and their use in knowledge-based systems significantly expands the scope of their current and future applications.

Advantages that apply to many grid-based tools, and Aquinas and ETS in particular, are presented below.

Improved Process Efficiency and Faster Knowledge Base Generation

Use of repertory grids can offer many advantages: rapid prototyping and feasibility analysis, vocabulary, solution and trait elicitation, interactive testing and refinement during knowledge acquisition, implication discovery, conflict point identification, expert system shell production, and generation of expert enthusiasm (Boose, 1986a). Experts may use these tools directly with little help from a knowledge engineer.

It is much easier for users to learn knowledge-based system concepts through using systems such as Aquinas than through reading books or attending classes (i.e., rules are automatically generated and used dynamically in consultations; new vocabulary is incrementally introduced).

ETS, still in use at Boeing, has been employed to build hundreds of small prototype systems. Alternative problem approaches may be explored with little expenditure of resources. Knowledge bases have been generated for expert system shells that contain over two-thousand rules. Many prototype systems have been built during the development of Aquinas; some contain thousands of judgments arranged in hierarchical rating grids. Small prototype systems containing several hundred ratings take only a few hours to build.

Improved Knowledge Base Quality

Aquinas offers a rich knowledge representation and reasoning environment. Hierarchies help the expert break down problems into component parts and allow reasoning at different levels of abstraction. Varying levels of precision are specified with multiple types of rating scales when needed. Multiple methods for combining uncertain information are available based on needed precision and convenience.

Knowledge from multiple experts can be combined. Users of Aquinas may receive dissenting as well as consensus opinions from groups of experts, thus getting a full range of possible solutions. Disagreement between the consensus and the dissenting opinion can be measured to derive a degree of conflict for a particular consultation. Grid tools can be used for cost-effective group data gathering.

Analytic capabilities in grid tools can help uncover inconsistencies and circularities in a growing knowledge base.

Better Knowledge Base Maintenance and Comprehensibility

In Aquinas, elicitation, structuring, analysis, and testing of knowledge is based on cases. When knowledge is updated, it is done so with respect to a specific case. Addition of new knowledge can be strictly controlled by the expert; the tendency for local changes to degrade other cases can be curbed.

With Aquinas, the expert builds and refines knowledge in rating grids and hierarchies - not directly in production rules, frames, or some other representation at a lower level of abstraction. As a result, knowledge is more compact, comprehensible, and easier to maintain.The growing collection of rating grids and case knowledge represents an important resource for building a variety of knowledge-based systems.

4.0 SUMMARY AND FUTURE WORK

In the future we plan to extend Aquinas in several areas:

Analysis of Cases and Automatic Knowledge Base Improvement. Aquinas' consultation mechanism is being improved to analyze cases in terms of the experts' expectations of performance. Trait sensitivity can be analyzed, improvements to grids may be suggested in terms of rating and weight changes, and new traits that improve performance can be synthesized (Shema and Boose, 1987).

Text Analysis. Simple forms of text analysis, such as a word distance measure used in Kitten and KSS0, can be used to identify seed solutions and traits for a domain. The expert would not need to start building repertory grids from scratch if domain-oriented texts were available for analysis.

Capturing Explanations. Templates will be used to capture comments from experts when making judgments (ratings, weights, traits from triads). Comments will be examined to help maintain the knowledge base and will be played back during consultations to provide explanations for the end-user. Applicable work on classifying explanations has been reported by Schank (1986) and Kass and Leake (1987).

Further Work in Acquiring Knowledge from Multiple Experts. In addition to Aquinas' current capabilities to analyze and use knowledge from multiple experts, we plan to implement the following features:

- Elicit knowledge in parallel (on-line) from several experts (Chang, 1986) in the same or different fields.
- Analyze subsumption and overlap of experts' knowledge in multiple fields (similar to SocioGrids in Planet).
- Support on-line structured negotiation among experts to solve problems or make recommendations.
- Discover the most important aspects of individual knowledge when compared to group knowledge (special cases).

Elicit and merge knowledge dynamically for use in on-line applications.

Eliciting Strategic Knowledge. We have performed experiments with Aquinas to elicit high level strategic knowledge. We took cases and listed them on the top of the grid, and then asked the expert, "What strategy might you use to solve two of CASE1, CASE2, and CASE3, but not the third?" Strategies were elicited as traits, and then the expert rated the applicability of each strategy to each problem case. This method tended to highlight the priorties of various procedures in solving the problem (Figures 12 and 13).

```
What strategy might you use to solve two of AUTO.ACCIDENT, MEDICAL.CALL,
and MOUNTAIN.RESCUE share, but not the third?
NEW STRATEGY** FIND OUT LOCATION

What is that strategy's opposite as it applies in this case?
NEW STRATEGY** LOCATION NOT ESSENTIAL

What is the name of a scale or concept that describes FIND.OUT.LOCATION /
LOCATION.NOT.ESSENTIAL?
NEW STRATEGY (CONCEPT)** LOCATION

What strategy might you use to solve two of MEDICAL.CALL, MOUNTAIN.RESCUE,
and MISSING.CHILD share, but not the third?
NEW STRATEGY** FIND OUT SEVERITY

What is that strategy's opposite as it applies in this case?
NEW STRATEGY** SEVERITY NOT ESSENTIAL

What is the name of a scale or concept that describes FIND.OUT.SEVERITY /
SEVERITY.NOT.ESSENTIAL?
NEW STRATEGY (CONCEPT)** SEVERITY
```

Figure 12. Triad comparison of cases is used to elicit strategies and procedures for the emergency medical situation advisor.

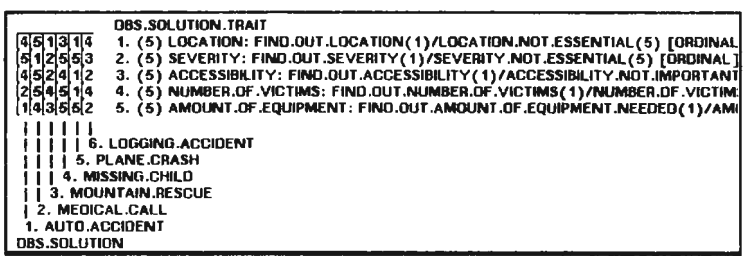


Figure 13. Rating grid of cases (columns) and strategies (rows) for the emergency medical situation advisor. For this problem, strategies are procedural priorities for finding out different types of information.

When this information is inverted (rows and columns are exchanged so that strategies become "solutions" and cases become "traits") the type of case can be used to select the appropriate strategy during consultations.

Knowledge Acquisition for Synthesis Problems. Attempts are being made to use grid tools to elicit knowledge for synthesis problems. For example, we are experimenting with allowing the user to put arbitrary computations in the cells of rating grids in Aquinas (spreadsheet-like computations, database access calls, functions that sample sensors). However, this will make the tool more difficult to use for these applications.

We are also planning to allow experts to specify complex interacting constraints between traits in Aquinas. This will help capture knowledge for portions of configuration and scheduling problems.

Eventually, we hope to blend together techniques from other tools (such as MDIS, Antonelli, 1983, and SALT, Marcus, 1987) to help elicit knowledge for aspects of design problems:

Acquisition of causal models and design constraints.

Selecting alternative design concepts based on competing criteria such as reliability, maintainability, cost, manufacturability.
Acquiring human expertise in validation and verification of designs.
Acquiring knowledge for design evaluation and change recommendation.
Acquiring knowledge for document generation and evaluation.
Capturing historical data in a "corporate memory" database to help solve future design problems.

Conclusion

Repertory grid-centered tools may be used in many contexts and are beneficial in a variety of application areas. Dimensions of usage were presented along with sample applications. The capabilities of these tools make them powerful decision aids, in and out of the traditional knowledge engineering process.

ACKNOWLEDGEMENTS

Thanks to Roger Beeman, Miroslav Benda, Jeffrey Bradshaw, William Clancey, Brian Gaines, Catherine Kitto, Ted Kitzmiller, Sandra Marcus, Art Nagai, Doug Schuler, Mildred Shaw, David Shema, Lisle Tinglof-Boose, and Bruce Wilson for their contributions and support. Aquinas and ETS were developed at the Knowledge Systems Laboratory, Advanced Technology Center, Boeing Computer Services in Seattle, Washington.

REFERENCES

Adams, J., Probabilistic Reasoning and Certainty Factors, in B. Buchanan and E. Shortliffe (Eds.), Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Reading, MA: Addison-Wesley, 1985.

Antonelli, D., The Application of Artificial Intelligence to a Maintenance and Diagnostic Information System (MDIS), in the *Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance*, Boulder, CO, 1983.

Benda, M., Baum, L. S., Dodhiawala, R. T., and Jagannathan, V., Boeing Blackboard System, in the *Proceedings of the High-Level Tools Workshop*, Ohio State University, October, 1986.

Boose, J. H., Personal Construct Theory and the Transfer of Human Expertise, in the *Proceedings of the National Conference on Artificial Intelligence (AAAI-84)*, Austin, Texas, 1984.

Boose, J. H., A Knowledge Acquisition Program for Expert Systems Based On Personal Construct Psychology, *International Journal of Man-Machine Studies*, Vol. 23, 1985.

Boose, J. H., Expertise Transfer for Expert System Design, New York: Elsevier, 1986a.

Boose, J. H., Rapid Acquisition and Combination of Knowledge from Multiple Experts in the Same Domain, *Future Computing Systems Journal*, Vol. 1, No. 2, 1986b.

Boose, J. H., and Bradshaw, J. M., Expertise Transfer and Complex Problems: Using Aquinas as a Knowledge Acquisition Workbench for Expert Systems, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1, 1987a.

Boose, J. H., and Bradshaw, J. M., AQUINAS: A Knowledge Acquisition Workbench for Building Knowledge-Based Systems, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, 1987b.

Bradshaw, J. M., and Boose, J. H., Decision Analytic Techniques for Knowledge Acquisition: Combining Situation and Preference Models Using Aquinas, *Proceedings of the 2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, October, 1987.

Chang, E., Participant Systems, *Future Computing Systems*, Vol. 2, No.1, 1986.

Clancey, W. J., Heuristic Classification, in J. Kowalik (Ed.), Knowledge-Based Problem Solving, New York: Prentice-Hall, 1986.

Diederich, J., Ruhmann, I., and May, M., KRITON: A Knowledge Acquisition Tool for Expert Systems, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1, 1987.

Diederich, J., Linster, L., Ruhmann, I., and Uthmann, T., A Methodology for Integrating Knowledge Acquisition Techniques, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, University of Reading, September, 1987.

Gaines, B. R., An Overview of Knowledge Acquisition and Transfer, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 3, *International Journal of Man-Machine Studies*, Vol. 26, No.4, 1987a.

Gaines, B. R., Rapid Prototyping for Expert Systems, in Oliff, M. (Ed.), *Proceedings of the International Conference on Expert Systems and the Leading Edge in Production Planning and Control*, University of Southern California, 1987b.

Gaines, B. R., and Shaw, M. L. G., New Directions in the Analysis and Interactive Elicitation of Personal Construct Systems, in M. L. G. Shaw (Ed.), Recent Advances in Personal Construct Technology, New York: Academic Press, 1981.

Gaines, B. R., and Shaw, M. L. G., Interactive Elicitation of Knowledge from Experts, *Future Computing Systems*, 1 (2), 1986.

Garg-Janardan, C., and Salvendy, G., A Conceptual Framework for Knowledge Elicitation, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, *International Journal of Man-Machine Studies*, Vol. 27, No.1, 1987.

Hinkle, D. N., The Change of Personal Constructs from the Viewpoint of a Theory of Implications, Ph.D. dissertation, Ohio State University, OH, 1965.

Kass, A., and Leake, D., Types of Explanations, Yale Computer Science Report YALEU/CSD/RR #523, March, 1987.

Kelly, G. A., The Psychology of Personal Constructs, New York: Norton. 1955.

Kitto, C., and Boose, J. H., Heuristics for Expertise Transfer: The Automatic Management of Complex Knowledge Acquisition Dialogs, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 2, *International Journal of Man-Machine Studies*, Vol. 26, No. 2, 1987a.

Kitto, C. M., and Boose, J. H., Selecting Knowledge Acquisition Tools and Strategies Based on Application Characteristics, *Proceedings of the 2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, October, 1987b.

Marcus, S., Taking Backtracking with a Grain of SALT, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, *International Journal of Man-Machine Studies*, Vol. 27, No. 1 , 1987.

Saaty, T. L., The Analytic Hierarchy Process, New York: McGraw-Hill, 1980.

Schank, R. C., Explanation Patterns: Understanding Mechanically and Creatively, Lawrence Erlbaum Associates, Englewood Cliffs, NJ, 1986.

Shaw, M. L. G., and Gaines, B. R., Techniques for Knowledge Acquisition and Transfer, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 2, *International Journal of Man-Machine Studies*, Vol. 27, No. 1, 1987.

Shema, D. B., and Boose, J. H., Refining Problem-Solving Knowledge in Repertory Grids Using a Consultation Mechanism, *Proceedings of the 2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, October, 1987.

Stewart, V., and Stewart, A., Business Applications of Repertory Grids, London: McGraw-Hill, 1981.

Decision Analytic Techniques for Knowledge Acquisition: Combining Information and Preferences using *Aquinas*

Jeffrey M. Bradshaw and John H. Boose
Knowledge Systems Laboratory, Advanced Technology Center, Boeing Computer Services
P.O. Box 24346, M/S 7L-64, Seattle, Washington 98124 USA (206) 865-3419

ABSTRACT

Many knowledge-based systems are prescriptive in nature. Their aim is not only to describe situations but also to recommend specific actions. Recommendations made by such systems depend on three things: the *alternatives* available, the *evidence* for consequences associated with the alternatives, and the *relative desirability* of these consequences. Research in knowledge acquisition has typically focused largely on the first two considerations: many approaches have been proposed for structuring and eliciting alternatives and for modeling uncertainty. However, relatively little effort has been made in the knowledge acquisition community toward an understanding of how to explicitly quantify the relationships between uncertainties and preferences. The field of decision analysis is concerned with the practical application of formal theories of probability and utility to the guidance of action. Decision analysis has been used for many years as a way to approach decisions with large amounts of uncertainty and multiple preference tradeoffs. This paper presents an approach to combining information and preference models that is being developed in *Aquinas*, an automated knowledge acquisition tool under development at Boeing Computer Services. The case for explicit preference models will be briefly presented. A view of the modeling of expertise will be given from the perspective of Personal Construct Theory. We will outline the approach of *Aquinas* to knowledge acquisition, and then present an application of this approach to an example that includes both preference and information components. Finally we will discuss the relevance of this to other approaches and outline future directions for research.

THE CASE FOR EXPLICIT MODELING OF PREFERENCES

Preference models and knowledge-based systems

Several recent papers have discussed the need for explicit preference models as part of the knowledge engineering process (e.g., Henrion, 1987; Henrion & Cooley, 1987; Keeney, 1986; Langlotz, Shortliffe, & Fagan, 1986; Wellman, 1986). In knowledge-based systems, preferences are typically treated implicitly and heuristically, making no provision for value structures differing from those built into the system.

In their discussion, Langlotz, Shortliffe, and Fagan (1986) cite an example rule from MYCIN (see Figure 1). This heuristic captures a physician's knowledge that tetracycline should be avoided for children because it may cause dental staining.

If:
1) The therapy under consideration is tetracycline
2) The age (in years) of the patient is less than 8
Then:
There is strongly suggestive evidence (.8) that tetracycline is not an appropriate therapy for use against the organism.

Figure 1: The MYCIN tetracycline heuristic, slightly simplified for illustration purposes.

Clancey (1983) gives a possible chain of four support rules for this heuristic (Figure 2). The first three inferences have to do with how one event influences the occurrence of the next. The fourth, however, is a compiled plan for action based on the inference chain. Langlotz et al. make the point that no matter how fine we break down a chain of reasoning, one rule in the chain will always recommend action based on the situation.

tetracycline in youngster
=> chelation of the drug in growing bones
=> teeth discoloration
=> undesirable body change
=> don't administer tetracycline

Figure 2: A justification for the tetracycline heuristic from Clancey (1983).

Because of the nature of heuristics, it is difficult to explicitly model the unique circumstances and tradeoffs that may justify an exception to the heuristic. What if, for example:

"- the infecting organism were resistant to all drugs except tetracycline?

- the only undesirable bodily change that tetracycline caused was minor intestinal distress?

-the probability of staining due to tetracycline was only 1 in 100? 1 in 1000?"

When tradeoffs are implicitly embedded within heuristics, it becomes impractical to ask, let alone answer such questions. Furthermore, fine tuning certainty factors as a way to represent unique tradeoff issues may have unpredictable effects on the rest of the knowledge base.

Advantages of explicit preference modeling Explicit modeling of preferences has several advantages:

1. Unique tradeoffs for a given situation can be addressed. If a particular child had an unusual resistance to dental staining, a change in treatment decision tradeoffs could be made cleanly without resorting to the tweaking of certainty factors.

2. The impact of a particular piece of evidence on a decision can be evaluated. By performing *sensitivity analysis*, we can examine whether a particular piece of evidence in favor of a diagnosis will have any real effect on the treatment decision.

3. The value of obtaining additional information can be ascertained. We can ask, "What is the most I should pay to conduct a test of the child's susceptibility to dental staining?"

4. The value of being able to control an uncertain variable can be determined. Imagine that there were a drug that effectively eliminated the possibility of dental staining due to tetracycline. How much is such a drug worth for a particular situation?

5. Risk attitude and time preference can be explicitly taken into account. In some medical situations, it may be worthwhile to model the patient's attitude toward risk before making a recommendation. Time critical situations may also require explicit modeling of unique circumstances.

6. System recommendations can be expressed in value terms. Sometimes it is important to for a system not only to recommended the favored treatment plan, but also to estimate its total cost in some meaningful unit of measurement.

We are beginning to implement methods for combining information and preference models in *Aquinas*[1], an automated knowledge acquisition tool under development at Boeing Computer Services (Boose and Bradshaw, 1987a, 1987b). The following section outlines our approach.

OUTLINE OF AN APPROACH

Elements of a Decision

Decision analysis has been used successfully for many years as a way to approach decisions with large amounts of uncertainty and multiple preference tradeoffs (Howard and Matheson, 1984; Keeney & Raiffa, 1976; Raiffa, 1968). Unfortunately, the expense and scarcity of decision analytic expertise has limited its application to decisions involving large amounts of resources. Recent efforts to remedy this problem by combining traditional decision analytic tools with knowledge-based systems have met with success in specific domains (Holtzman, 1986, 1988). We have found many of the concepts of decision analysis to be of practical use in the automation of knowledge acquisition.

From the point of view of decision analysis, there are three major kinds of knowledge that decision makers bring to the problem solving context: *information* ("What do I know?"), *preferences* ("What do I care about?"), and *alternatives* ("What can I do?"). In the MYCIN example given above, the information consists of the knowledge a physician relating symptoms and treatments to diseases; the preferences consist of factors that determine the desirability of an alternative such as cost, effectiveness, or risk; and the alternatives consist of the various possibilities for treatment. We extend the decision analytic view to include specializations of the three kinds of knowledge (see Figure 3). [2]

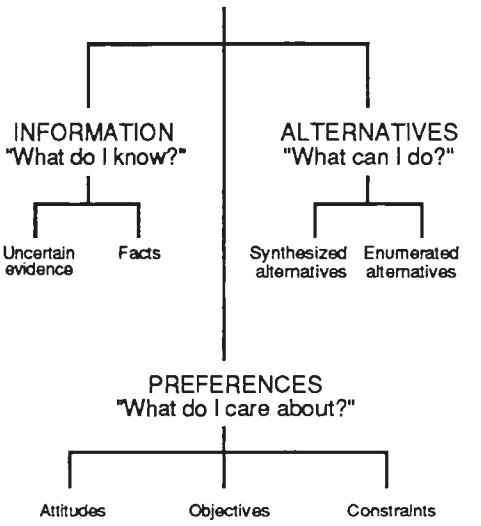


Figure 3: A decision maker brings information, alternatives, and preferences to the problem-solving context.

Information. Information may be of two types: *uncertain evidence* which consists of statements that are believed to be true with some probability, and

facts which are statements that are believed with certainty. Facts are, of course, simply a special case of uncertain evidence where the mapping onto the space of probability distributions happens to be concentrated on a single point. However by asserting that something is a fact, we can sometimes formulate a problem with additional clarity and computational efficiency.

Preferences. We are motivated by the strengths of our preferences to make a careful choice among alternatives. If we really didn't prefer one set of consequences to another, any effort we put into making a decision would be meaningless. In structuring preferences, it is useful to distinguish between *attitudes, objectives,* and *constraints.*

> 1. *Attitudes* consist of desires about things like time (e.g., the desire to receive good outcomes sooner rather than later) and risk (e.g., the desire to pursue a possibly less profitable policy in order to avoid risk).
>
> 2. *Objectives* are related to the significant positive and negative consequences of the alternatives that a decision maker wishes to maximize or minimize. In decisions with multiple objectives, methods of quantification and joint measurement (i.e., commensuration) must be found so that tradeoffs can be made between them. For example, in medical decision making we must sometimes consider not only the success of the major treatment outcome, but also factors such as discomfort, lifestyle, and length of life.
>
> 3. *Constraints*, like objectives, are related to the consequences of the alternatives, but differ in that they must inviolably be satisfied. Constraints may be matters of definition (e.g., "There are three million dollars budgeted for the project this year.") or of principle (e.g., a moral belief that precludes considering alternatives such as murder). In reality, few constraints are truly hard — for example, workers may be willing to violate the constraint of an eight-hour work day given sufficient overtime compensation.

Alternatives. Alternatives must either be *synthesized* as a result of evaluating the information and preferences or may, where practical, be *enumerated.* Although this paper will discuss only problems where solutions may be conveniently enumerated, it is important to note that the traditional distinction between analysis and synthesis problems arises not so much because of some fundamental incompatibility in how they may be conceptualized, but rather because their differences may be exploited by divergent problem solving approaches (Clancey, 1984, 1986).

As a final observation, we note that different types of problems require different amounts of emphasis on modeling these kinds of problem-solving knowledge. In the types of design and configuration problems that SALT (Marcus, 1987) has addressed, there has been a strong requirement for the modeling of preferences in the form of constraints, objectives and procedures for obtaining them. For these problems, however, there has been little need to model uncertain information. On the other hand, in a purely diagnostic or interpretive task, where no recommendations for action would be made, there is great value in modeling uncertain information but no compelling need to model preferences. Decision problems, such as the ones for which *Aquinas* attempts to acquire models, generally require both information and preference modeling in order to recommend alternatives.

Personal Construct Theory and Knowledge Acquisition

A growing number of knowledge acquisition tools incorporate methods derived from Personal Construct Theory (Kelly, 1955; Gaines and Shaw, 1981). Such tools include the Expertise Transfer System (ETS, Boose, 1984, 1985, 1986), Planet (Gaines and Shaw, 1986), Aquinas (Boose and Bradshaw, 1987a, 1987b; Kitto and Boose, 1987), FMS Aid (Garg-Janardan and Salvendy, 1987), Kitten (Shaw and Gaines, 1987), Kriton (Diederich, Ruhmann, and May, 1987; Diederich, Linster, Ruhmann, and Uthmann, 1987), KSSO (Gaines, 1987a, 1987b) and others. The relevance of Kelly's theory to knowledge acquisition is due to his rich characterization of the efforts of individuals to actively anticipate and control their environment. His analysis from the perspective of how individuals anticipate and represent events resulted in a unique theory of how personal models of the world are created and maintained. Consistent with this view, Weimer writes:

> "There is no difference in kind between the scientist inferring the most esoteric theory of reality, on the one hand, and the simplest organism's inferring the presence of food or danger in its environment. In both cases the fundamental *activity* of the nervous system is *classification* (or abstraction) and the fundamental *function* of the nervous system is *modeling* (of the environment)" (1975, p. 455)

Gaines (1987a, 1987b) has extended the ideas of Kelly and others to propose an epistemological hierarchy accounting for the main components of modeling systems and their inter-relations (Figure 4):

"• At level one, *constructs* are those distinctions that the particular modeling system makes, a language for describing of the world;
- At level two, *data* are descriptions of actual case histories in terms of the constructs, an account of experiences of the world;
- At level three, *hypotheses* are the means of regenerating particular case histories from generalized accounts, rationalizations of the world (often called *models*);
- At level four, *analogies* are similarities between differently generated generalized accounts, correspondences between models;
- At level five, *abstractions* are accounts of a wide range of models, underlying foundations of analogies;
- At level six, *principles* are systemic foundations for abstraction, accounts of abstractions." (Gaines, 1987b)

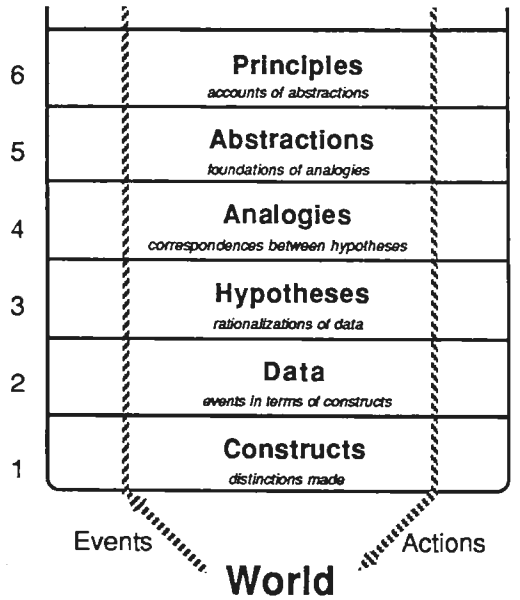


Figure 4: Gaines' (1987b) construction hierarchy of a person modeling the world.

The notion of a process involving *information, preferences,* and *alternatives* is consistent with this hierarchy. Looking at figure 4, we notice that the fundamental process in the model is the flow of information about events upward through the hierarchy, and the flow of preference down the hierarchy resulting in action:

> "Modeling may be evaluated by the degree to which a level accounts for the information flowing through it. This may be measured as the *surprise* at an actual event in relation to one predicted, and the expected surprise is a form of entropy (Gaines, 1977)...
>
> The formal theory of modeling is one in which models are selected at each level down the hierarchy to minimize the rate at which surprise is passing up the hierarchy... The criteria for model selection independent of the data... in its most general form is just one of preference. Hence the general modeling schema is one in which surprise flows up the hierarchy and preference flows down." (Gaines, 1987b).

As an illustration of how models are selected to minimize surprise, consider that the distinctions that form the basis of our mental models do not arise as a direct consequence of the intrinsic properties of things in themselves. Rather, they are formed as a result of *surprising events* or *breakdowns:*

> "As I sit here typing a draft on a word processor... I think of words and they appear on my screen. There is a network of equipment that includes my arms and hands, a keyboard, and many complex devices that mediate between it and the screen. None of this equipment is present for me except when there is a breaking down. If a letter fails to appear on the screen, the keyboard may emerge with properties such as 'stuck keys.' Or I may discover that the program was in fact constructed from separate components such as a 'screen manager' and a 'keyboard handler,' and that certain kinds of 'bugs' can be attributed to the keyboard handler...
>
> For me, the writer, this network of objects and properties did not exist previously. The typing was part of my world, but not the structure that emerges as I try to cope with the breakdown..." (Winograd & Flores, 1987)

In response to a surprising event, new distinctions emerge and new possibilities for action are created. In the words of Hayek:

> "Our knowledge of the phenomenal world raises problems which can be answered only by altering the picture which our senses give us of that world" (Hayek, 1952, in Weimer, 1975, p. 461).

The incorporation of these new distinctions into a model may lead to better anticipation of future events:

> "Our expectations of the future are determined by our constructive memories: hence our theory of memory is *ipso facto* our theory of 'expectation' which in turn is *ipso facto* what guides our concept formation, which in turn is what guides scientific life."(Weimer, 1975, p. 480)[3]

Modeling information and preferences in *Aquinas*

Figure 5 illustrates the modeling process of Aquinas. A fuller explanation of this figure will be available in the final version of this paper.

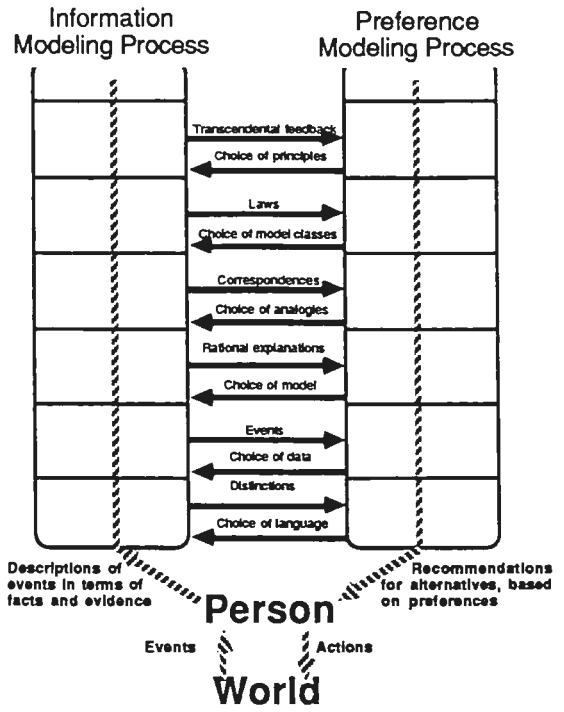


Figure 5: Modeling process of Aquinas.

We will follow four steps in building a model for an example decision problem:

1. Build an initial model of preferences and alternative actions.

2. Perform sensitivity analysis and look at value of information and value of control.

3. Build models of information relevant to sensitive variables.

4. Test and refine the combined model.

MODELING A SIMPLE DECISION: THE TWO DOGS PROBLEM

What is knowledge acquisition?

The *two dogs problem* begins in this way:

> I had two dogs. They had to go into quarantine for six months. It was a hot summer. At the beginning of that time, they thought that all wasps were flies and at the end of that time, they thought that all flies were wasps. [4]

The question to be asked is: Had the dogs learned anything during their quarantine? If so, what is the knowledge that was acquired?

On the one hand, it could be argued that they had learned nothing, since they had just substituted one overgeneralization ("No flying insects sting") for another ("All flying insects sting"). There had been no increase in the dogs' ability to discriminate between stinging and non-stinging insects.

We are uncomfortable with this formulation, however, because the dogs had presumably changed their behavior as a result of their experience. If they had begun to avoid all flying insects we could suppose that "learning" of some sort had taken place. It could be argued that the dogs' new pattern of avoiding all insects was a better strategy than their previous one even if there was no gain in information about the insects that was involved in that improvement.

We would argue then that learning by individuals and by systems is to be measured, not as an increase in "knowledge," but as an increase in the expected value of the best available alternative within a model for a given situation.

If we accept this pragmatic definition of knowledge acquisition, we realize that values are an inherent part of all decision processes and are, at least implicitly, a part of all prescriptive knowledge-based systems. A metric for evaluating the improvement in the dogs' decisions as a result of learning must incorporate not only the degree to which they can accurately identify the stinging potential of an insect, but also the preference tradeoffs between the effort involved in being overcautious and the pain involved in being too careless.

We will now build a representation of the dogs' preferences for alternative actions and their information about the stinging potential of various kinds of bugs in terms of rating grids and directed graphs. This representation should not merely reflect

what they currently do (i.e., *descriptively*), but more importantly give them insight into how they might improve their models. We will first look at the dogs' alternatives and preferences.

Building a preliminary model of the dogs' alternatives and preferences

The purposes of modeling the dogs' preferences is to discover key tradeoff variables relating to the alternatives. By modeling preferences explicitly and separately from evidence, we will be able to see the direct effects of varying preference model parameters without affecting the evidential structure of the model.

We can represent preferences for avoiding and not avoiding flying insects for one of the dogs by the rating grid in figure 6. The alternatives (also called *solutions*) **Avoid** and **Don't-avoid** relate to the grid columns and the preference variables (also called *traits*) **Level-of-effort** and **Get-stung?** relate to the rows. The *ratings* for **Level-of-effort** reflect the fact that avoiding the insects required more effort than not avoiding them and the ratings for **Get-stung?** reflect the dog's judgment that avoiding the insects was 100% effective in preventing stings, while not avoiding them resulted in a 50% chance of getting stung. A description of many of the methods for constructing and analyzing grids in *Aquinas* may be found in (Boose & Bradshaw, 1987a).

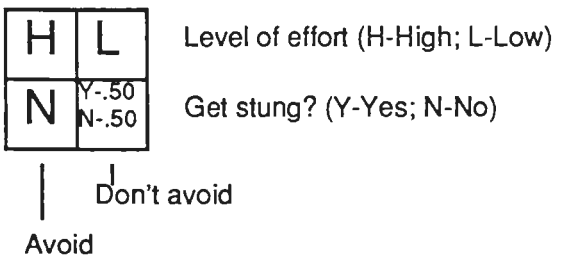


Figure 6: Rating grid representation of the one of the dog's preferences.

Once an expert has entered a set of solutions, traits, and ratings the grid can be used to make decisions. *Trait weights* elicited from the expert represent the relative costs of getting stung (.7) vs. making efforts to avoid the insects (.3). Since the traits are binary, we can assign a utility of 1 to the preferred values of each trait (Low, No) and a utility of 0 to the non-preferred ones (High, Yes).

Assuming additivity and linearity, the strength of a decision maker's preferences for an alternative is computed by taking the product probability of each value of the rating distribution for each trait with its corresponding utilities and the trait weight of that trait, and summing over all the traits, i.e.:

Avoid: .3((1 x 0)+(0 x 1))+.7((1 x 1)+(0 x 0))= .7
Don't Avoid:.3((1x1)+(0x0))+.7((.50x1)+(.5x0))= .65

It is often useful to express the expected values of alternatives in a common unit such as dollars. To do this, we ask experts to tell us how much they would pay to magically transform the worst set of outcomes into the best set. The dog might reply that he would pay $50 to go from a situation where he had been stung despite his best efforts to avoid the insect to a situation where he would not be stung and would have to make no effort. Consistent with our assumptions of additivity and linearity, *Aquinas* assigns a value of $35 to the situation where he is not stung and the level of effort is high, and $15 to the situation where he is stung and makes no effort.

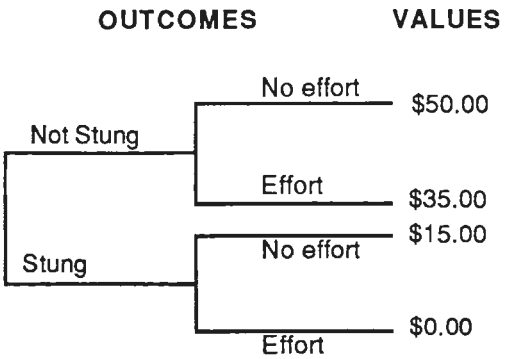


Figure 7: Utilities can be converted to dollars by asking experts to tell us how much they would pay to transform less desirable outcomes into more desirable ones.

When the least preferred value of a trait is not "too bad" and the spread between the most and least preferred is not "too great", a decision maker's utility function may be almost linear. In other cases, an individual's *risk attitude* should be taken into account. As an illustration, consider the fact that many people are *risk preferring* or *risk neutral* for small amounts (e.g., a double or nothing gamble on a few dollars) whereas they would be *risk aversive* for larger amounts (e.g., a double or nothing gamble involving a billion dollars).[5] Where risk attitude is an important consideration, the worth of alternatives should be expressed in terms of *expected utility* rather than *expected value*. For this problem, we will not assess risk attitude.

It is often useful to decompose preference models hierarchically. For example, if the dog found it difficult to give ratings for a global **Level-of-effort**, we could break that trait down to form a sub-grid where the traits would be various components of effort and the solution elements would be **Avoid/Don't-avoid**. The unrated grid cells at the

higher level could then be filled in automatically by *Aquinas* from information in the sub-grid.

Appraising the preference model

Model appraisal can be used to accomplish two purposes: 1. To discover whether the model can be simplified by eliminating variables or alternatives that do not affect the final decision, and, 2. To discover how changes in values of one or more variables affect the selection of alternatives so that areas where more careful modeling is needed can be identified. For the sake of brevity, we will only illustrate a single example of the second purpose here.

Sensitivity analysis. The one uncertainty in our preference model is whether or not a particular insect will sting when no effort is made to avoid it. In sensitivity analysis, we repeatedly calculate the expected value for each of the alternatives while varying the probability of each outcome. Figure 8 shows how the expected value of the alternatives varies with the probability of getting stung.

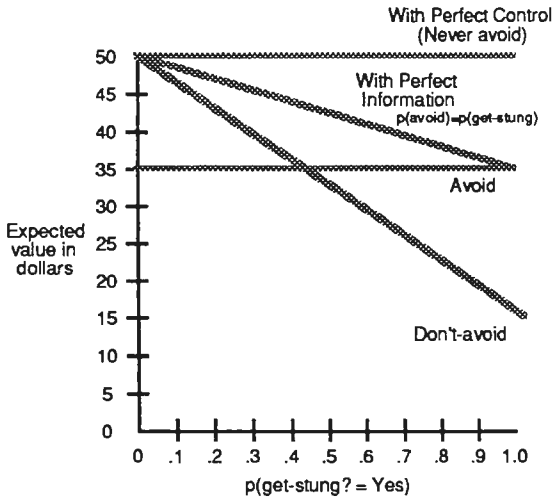


Figure 7: Graph showing how the probability of getting stung affects the value of the alternatives, the value of control, and the value of perfect information.

In examining the relative positions of the Avoid and Don't-avoid alternatives at the 0.5 level, we see that the Avoid alternative slightly dominates with our current state of information. By looking at where these two strategies cross, we can conclude that if the dog could reliably predict that the chance of getting stung were less than .43, it would be just as well for him to rest easy in the shade as to take evasive action:

Avoid:
.3((1 x 0)+(0 x 1))+.7((1 x 50.00)+(0 x 0)) ≈
Don'tAvoid:
.3((1x50.00)+(0x0))+.7((.57x50.00)+(.43x0))

Since reducing the dog's uncertainty about getting stung to less than .43 will result in our choosing a higher valued alternative, we will examine this variable more carefully by determining the value of control and the value of information.

Value of control. The line labeled "With perfect control" represents the expected value of the decision if the dog could somehow eliminate the possibility of stinging insects by taking preventive action. He could, for example, build a closed shelter for himself where no insects could enter. In such a case, he would never have to choose the Avoid alternative. The difference in expected value between having perfect control and the best alternative for a given probability is called the value of control and can be used as an upper bound on what the decision maker should pay to increase control. In our example, the dog should spend no more than \$50.00-35.00=15.00 to build such a shelter if the probability of getting stung is greater than .43. The value of control will be less than \$15 when p(get-stung)<.43.

Value of information. The line labeled "With perfect information" represents the expected value if the dog could know with certainty whether the insect would sting or not before he made his decision to avoid or not avoid it. Since he cannot control the situation, the best he can expect is that he have to choose the Avoid alternative with a relative frequency corresponding (in the long run) to the probability of getting stung. The value of perfect information analysis tells us that if the dog could buy a wasp detector that performed with 100% accuracy, he should pay no more than \$42.50-35.00=7.50 for it, given our current state of information.

Imperfect information can also increase expected value. For example, we can determine how much the dog ought to pay for a wasp detector that had only limited reliability.

On the basis of our appraisal, we decide to model what the dog knows about the stinging potential of insects more carefully.

Building a model of the dog's information

The dog determines that two factors are relevant to the likelihood of an insect stinging:

1. Probability that the insect *can* sting.

2. Length of time the insect stays close by, given that it can sting.

In thinking about the first factor, the dog decides that the best evidence he has for determining whether an insect can sting is its appearance. So we will ask him

to think about some of the differences between the insects that sting and those that don't. Figure 8 is a graphical representation of the results. The two types of insects have been assigned relative positions on three dimensions: *size*, *position*, and *mood*. Uncertainty is represented by the width of the stinger and non-stinger blocks along a particular dimension, indicating that we are unsure of where the exact value lies.

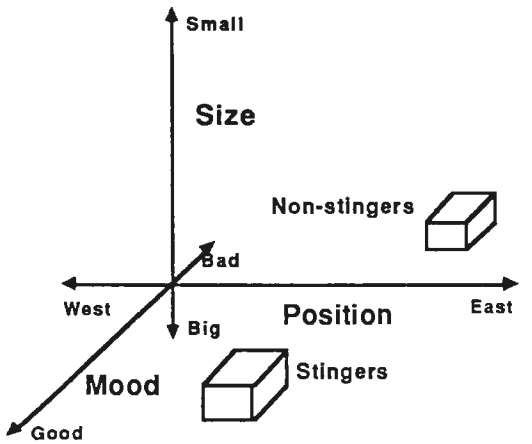


Figure 8: Dog's view of uncertain evidence relating to the insects' appearance. The axes are orthogonal to one another, reflecting the dog's initial belief that the dimensions are independent.

From the figure, we can see some differences: Stingers are to the west of Non-stingers and Stingers, although bigger than Non-stingers, seem generally to be in a better mood. Note that the uncertainty about the mood of the insects is greater than the uncertainty about their position, which is what we would expect.

There are three ways we can go about quantifying these beliefs:

1. Derive probabilities from data.
2. Ask experts to make probability assignments.
3. Combine data with expert judgment.

Each of these methods is illustrated below.

Deriving probabilities from data
Given a hypothesized set of differences between stinging and non-stinging insects, we can perform an experiment. The dog predicted that any new stingers would be, on the average, relatively colder, in a better mood, and further east than the non-stingers. Here are our results after observing eight insects, four of which stung:

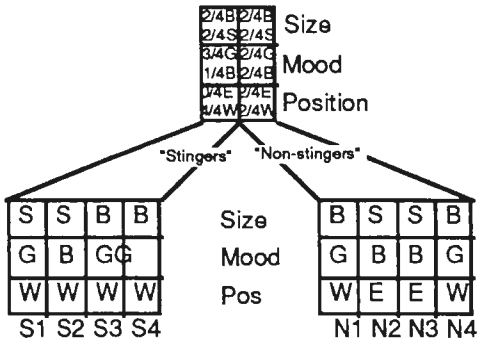


Figure 10: Results of observing eight insect, four of which stung and four of which did not.

We can see by inspection that the data set is very noisy. A non-probabilistic classification algorithm would be unable to discriminate between the classes.

Just as preferences can be broken down hierarchically within *Aquinas*, so solution elements can be reasoned about with grids at different levels of abstraction. The top grid in Figure 9 derives its rating value distributions for each column from the grids of examples shown. This grid in turn could be used in deriving distributions for the element "flying insects" in a possible higher level grid. (*Aquinas* can also use information in higher level grids to derive missing values in lower ones.)

Limitations of common methods for learning from data. Many approaches have been proposed for inducing generalizations from data. We emphasize three limitations of most empirical learning systems:

1. Inability to find or make use of relationships that are only statistically true. Many empirical learning systems depend on procedures that look for relationships that are always (or nearly always) true and attribute any other findings to noise. Unfortunately, many knowledge-based systems are targeted for domains where causal mechanisms are incomplete or poorly understood. Such domains are dominated by probabilistic rather than deterministic relationships. Recent advances in probabilistic modeling that demonstrate its practicality in knowledge-based systems have been well-documented in the literature (Cheeseman, 1985, 1986; Henrion, 1986; Pearl, 1985; Spiegelhalter, 1986a, 1986b).

2. Fixed undependence assumptions. Early attempts to model uncertainty probabilistically such as PROSPECTOR (Duda, & Reboh, 1984) and the original singly-connected Bayes' network formulation (Kim & Pearl, 1983) relied on the conditional independence assumption for reasons of computational tractability. Fuzzy set operators, on the other hand, assume the maximum possible

correlation between items of evidence. In some cases, these fixed assumptions can have a significant negative impact on reasoning (Wise & Henrion, 1986).

3. Overfitting the model to the data. Even when inductive learning systems take probabilistic relationships into account, they may lack appropriate significance testing to assure that they are not "fitting the noise". Various tree-pruning heuristics or stopping rules have been devised for systems such as (Quinlan, 1983, 1986) but no single agreed upon criterion exists.

Approach to learning from data. We are implementing a method for extracting significant joint probabilities from data that addresses each of these limitations. The reader is referred to (Cheeseman, 1984, 1987) for a complete description.

Within this framework, we can regard an inductive generalization as a way of representing more briefly the information contained in the set of examples. The generalization we make will represent a tradeoff between our desires for accuracy and completeness on the one hand ("minimizing surprise") and our desire for parsimony on the other hand ("maximizing preferences"). We would also like to avoid incorporating into our model the random characteristics of a particular data set (i.e., overfitting).

By taking into account the class to which the insect belongs and the relative frequency of each trait value for that class we can compute expected probability values for combinations of trait values under a generalized independence assumption (maximum entropy). Where the observed frequencies for the combinations of trait values do not differ significantly from the maximum entropy probabilities, we can avoid having to represent the dependencies explicitly in a joint distribution table. Our best estimate in such instances can be efficiently computed because we can assume conditional independence. Where such differences are significant[6], we can represent the observed values as an explicit constraints on the maximum entropy calculation.[7]

This procedure allows us to estimate joint probabilities to precisely the accuracy justified by the noise in the data. The noise is due to the fact that the data is only a sample. If new data provides additional support for the significance of a value not previously found significant, the generalization can be easily revised.

Although this approach saves us from specifying a full joint distribution and neatly solves the problem of overfitting, it can lead to complex calculations if there are a large number of significant probabilities. Shastri (1985) suggests that the most effective use of information about significant correlations between trait values is in the creation of new concepts. For example, an effective use of the information "Most big non-stingers are in a good mood," would be to create the new categories of "Big-non-stingers" and "Small-non-stingers" and attach the appropriate information on mood and position.

The example we have just discussed is an instance of "supervised learning": that is, the examples given to the system had been pre-classified as being either stingers or non-stingers. *Aquinas* can also "learn" in an unsupervised mode by performing a hierarchical classification of the examples using cluster analysis. The current algorithm is based on trait similarity; however we are investigating other approaches based on information theoretic formulations similar to those described above (Boulton & Wallace, 1973; Cheeseman, 1987).

Asking experts to make probability assignments

In the absence of data, the best we can do is to ask experts to make probability assignments directly. Figure 11 shows the dog's judgments of the probability of the values of each trait for stinging and non-stinging insects.

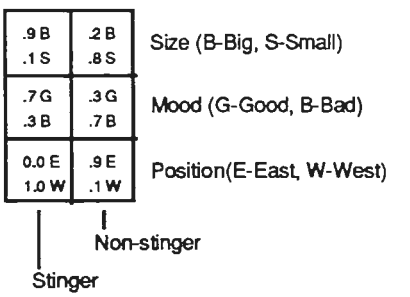


Figure 11: Probabilities obtained directly from the dog. The prior probabilities for Stinger and Non-stinger were also obtained (not shown).

Issues in direct assessment. It is not always easy to get experts to make numerical assessments of uncertainty (Eshelman et al., 1987). In domains with a strong causal model, where there is little uncertainty or conflicting evidence, all reasonable uncertainty inference procedures (including qualitative ones) should perform reasonably well, and in fact the simplest method is to be preferred. When accurate numerical assessments are required, there are two features of the present approach that make the task somewhat less difficult.

First, by asking for assessments in terms of probabilities rather than other numbers like certainty

factors, anchor points can be clearly defined for the expert. Because there is an unambiguous meaning to the numbers we can make exact mappings between graphical representations such as probability wheels and histograms in *Aquinas* and probabilities. Equivalent mappings for other numerical representations are problematic.

Second, we find that experts respond much more readily to a "backward approach to assessment". Many rule-based systems represent knowledge in the form:

> IF <evidence> THEN
> <hypothesis>, CF,

where CF is a measure of change in belief for the hypothesis given the evidence. (Schachter & Heckerman, 1987) have shown that while the direction of usage of evidence is usually from evidence (observable) to hypothesis (unobservable), experts are usually much more able to give assessments in the opposite direction: IF <hypothesis> then <evidence>. When filling in a rating grid such as the one in figure 11, experts are doing just that: they provide numbers representing the evidence for each trait, given the solution.

Obtaining information about dependencies. If the expert gives us only the assessments in the grid of figure 11, we can say nothing about dependencies between traits. If the expert is unwilling to provide additional information, the best we can do is to assume maximum entropy and compute joint probabilities as if the traits were independent. If the expert can provide information about dependencies between traits, we generate the grid shown in figure 12. We call this a hypothetical example grid because the task is for the expert to estimate the probabilities of occurrence of various combinations of trait values for a hypothetical example set[8]. The default probabilities are the product of the probabilities for the corresponding column values of each trait in figure 11. Experts adjust these probabilities until they are satisfied that the relative proportions of each combination of values are consistent with their experience[9]. The significance of the correlations between traits can be evaluated in the same way as it was done when we had real observations available.

Stingers

B	S	B	S	B	S	B	S	Size
G	G	B	B	G	G	B	B	Mood
E	E	E	E	W	W	W	W	Position
0	0	0	0	.67	.07	.27	.03	

Figure 12: Hypothetical example grid for Stingers. The probabilities shown are the product of the probabilities for the corresponding column values of each trait in Figure 11.

Combining data and expert judgments
The approach we have described makes it convenient to combine data with subjective assessments. To do this, we ask experts to express their probabilities as fractions, with the denominator representing an estimate of the total number of cases on which the judgment is based and the numerator representing the fraction of the cases with a particular combination of characteristics (Spiegelhalter, 1986a, 1986b). Then we combine these fractions with those from our data (Figure 13). Again, the same significance test for the joint probabilities can be applied.

Stingers

	B	S	B	S	B	S	B	S	
	G	G	B	B	G	G	B	B	Mood
	E	E	E	E	W	W	W	W	Position
Expert	0/30	0/30	0/30	0/30	19/30	2/30	8/30	1/30	
Data					2/4	1/4		1/4	
Combined	0/34	0/34	0/34	0/34	21/34	3/34	8/34	2/34	

Figure 13: Combining data and expert judgments.

(Hink and Woods, 1987) have noted a number of interesting features of this approach:

> "First, as the system gains experience, its measures of association converge toward the true values (that is, values supported by the objective observations). Second, with initially confident subjective estimates (that is, large cell values), this convergence takes longer. Third, this methodology naturally incorporates several of the psychological techniques for unbiasing subjective assessments of uncertainty, such as focusing on disconfirming conditions and using concrete measures (for example, frequencies rather than probabilities). Finally, limitations of precision with human judgment are overcome; precision is limited only by the amount of data acquired."

Accumulating data can also be used as a comparison with the original assessments in order to suggest changes to the knowledge base when the system is too often "surprised" by what it sees. One measure of such discrepancies is the 'Brier score' which is equivalent to the squared predictive error (Spiegelhalter, 1986a). If an insect turned out to be a stinger when its current probability was .75, the Brier score would be:

$(1-.75)^2 = .0625$

However, if the insect turned out not to be a stinger, the score would be:

$(0-.75)^2 = .5625$

registering increased surprise. If a series of n events were recorded, the total Brier score would be $B = \Sigma (1-p)^2$. The expected Brier score if the system were reliable would be $E_0(B) = \Sigma p_i (1-p_i)$ with variance $V_0(B) = \Sigma (1-2p_i)^2 p_i(1-p_i)$. Thus $[B-E_0(B)]/V_0(B)^{1/2}$ is the standardized test statistic for calibration.

Linking the information and preference models

Figure 14 shows a network-style representation of the model combining information and preference variables. The two preference variables, **Get-stung?** and **Level-of-effort** directly determine which alternative action is recommended. Whereas **Level-of-effort** is given static rating values by the expert, **Get-stung** will derive its values dynamically by querying the linked information model. The terminal value in the information model is derived by combining evidence about the **Length-of-time** and whether or not the insect appears to be a **Stinger** or **Non-stinger.** The **Stinger/Non-stinger** node in turn derives its values from its predecessors. If no evidence is available, the best estimate it can provide is the prior probability that the insect belongs to either class.

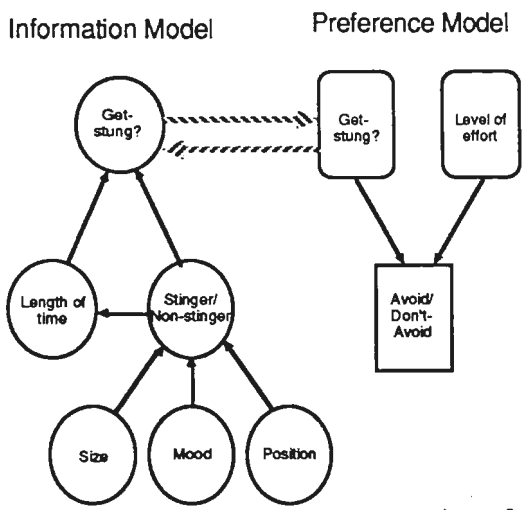


Figure 14: A network-style representation of the information and preferenc models.

Our goal in building the information model was to reduce the dog's uncertainty of an insect's stinging potential by providing a framework in which additional relevant information could be obtained. We saw as a result of value of information analysis that additional information could sometimes increase expected value.

But we have ignored the cost of asking and answering questions. Sometimes it may be easier to just avoid the insects than to have to inspect them in detail. Besides, gathering additional information takes time, and the longer the delay the greater the likelihood of getting stung (if the insect is a stinger).

We can represent the alternative of answering questions about the insects to gain additional information as a third column in the preference grid (see Figure 15). The rating values for **Level-of-effort** will change dynamically depending on which question is being asked, since some questions may be more difficult to answer than others. The rating values for **Get-stung?** will also be determined dynamically as a function of the current state of information plus the information expected as a result of getting an answer to that question.

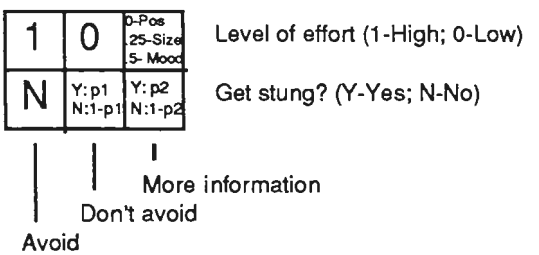


Figure 15: Preference rating grid incorporating the alternative of getting more information. p1 = min(.99, $p(\text{stingerle}_c)$ x $\text{length-of-time}_c/10$). p2 = min(.99, $p(\text{stingerle}_c)$ x $(\text{length-of-time}_c+\text{length-of-time}_q)/10$ x $H(\text{stingerle}_c, q)$.

The difference between the expected value of choosing **More-information** and the expected value of the next best alternative (either avoiding or not avoiding) is the expected value of information for that question. We will represent the cost of delaying an avoidance tactic in order to get more information as the increase in probability that the insect will sting, given the length of time it has been in range and the length of time it will take to answer the question. The length of time to answer a question will be computed as $\text{length-of-time}_q = .1 \text{ x } \text{Level-of-effort}_q$. The probability that the insect will sting given elapsed time is p(will-stinglength-of-time, stinger) = min(.99, length-of-time/10). The probability of being stung given the **Don't-avoid** alternative is computed as p1 = min(.99, $p(\text{stingerle}_c)$ x $\text{length-of-time}_c/10$). The probability of being stung given the **More-information** alternative is p2 = min(.99, $p(\text{stingerle}_c)$ x $(\text{length-of-time}_c+\text{length-of-time}_q)/10$ x $H(\text{stingerle}_c, q)$. $H(\text{stingerle}_c, q)$ represents the expected information gain from an answer to the question. This will be discussed below. The trait **Level-of-effort** has been made more sensitive by changing its legal rating values from High/Low to a range of 1-5.

Running a test consultation
After asking a series of questions to set up the consulation context, *Aquinas* evaluates the combined model to produce a recommendation. If the preferred alternative is to **Avoid**, the recommendation is made and the consultation is finished. If the preferred alternative is **Don't-Avoid,** the recommendation is made but the system monitors changes to rating values over time until a new alternative is recommended. When *Aquinas* recommends **More-information**, the system will ask a question about the insect's size, mood, or position, then recompute to discover the best alternative, given the new evidence. This cycle will continue until we have run out of questions or until the system makes a final decision to avoid or not avoid the insect.

We have seen how information about information cost can be incorporated into *Aquinas'* question asking strategy. As mentioned above, we also compute a measure of expected information gain, $H(\text{stinger}|e_C, q)$ that gives us some measure of the potential usefulness of a question. Users of *Aquinas* could design appropriate cost/benefit strategies for a particular application by combining their cost function with the measure of expected information gain.

We compute the entropy of classification, $H(Cla_j)$, of each attribute, a_j, by the expression:

$$H(Cla_j) = - \Sigma_{i=1 \text{ to } N} p(c_i|a_j) \log_2 p(c_i|a_j)$$

where $p(c_i|a_j)$ is the probability that the class is c_i when the attribute is a_j (Quinlan, 1983). The trait with the lowest value for H is the one that should be asked about first. Our computations of H will be based upon the example set in figure 10:

SIZE
H(non-stinger|small) = - (2/4 x $\log_2$(2/4)) - (2/4 x $\log_2$(2/4)) = 1.0
H(non-stinger|big) = - (2/4 x $\log_2$(2/4)) - (2/4 x $\log_2$(2/4)) = 1.0
H(non-stinger|size) = (4/8 x 1.0) + (4/8 x 1.0) = 1.0

MOOD
H(non-stinger|good) = - (2/5 x $\log_2$(2/5)) - (3/5 x $\log_2$(3/5)) = 0.972
H(non-stinger|bad) = - (2/3 x $\log_2$(2/3)) - (1/3 x $\log_2$(1/3)) = 0.916
H(non-stinger|mood) = (5/8 x 0.972) + (3/8 x 0.916) = 0.951

POSITION
H(non-stinger|least) = - (2/2 x $\log_2$(2/2)) - (0/2 x $\log_2$(0/2)) = 0.0
H(non-stinger|west) = - (2/6 x $\log_2$(2/6)) - (4/6 x $\log_2$(4/6)) = 0.916
H(non-stinger|position) = (2/8 x 0.0) + (6/8 x 0.916) = 0.687 ***

Position has the highest expected value of information. Next, we compute the entropy of classification for Size and Mood given position:

SIZE
H(non-stinger|west, small) = - (0/2 x $\log_2$(0/2)) - (2/2 x $\log_2$(2/2)) = 0.0
H(non-stinger|west, big) = - (2/4 x $\log_2$(2/4)) - (2/4 x $\log_2$(2/4)) = 1.0
H(non-stinger|west, size) = (2/6 x 0.0) + (4/6 x 1.0) = 0.667 ***

MOOD
H(non-stinger|west, good) = - (2/5 x $\log_2$(2/5)) - (3/5 x $\log_2$(3/5)) = 0.972
H(non-stinger|west, bad) = - (0/1 x $\log_2$(0/1)) - (1/1 x $\log_2$(1/1)) = 0.0
H(non-stinger|west, mood) = (5/6 x 0.972) + (1/6 x 0.0) = 0.807

Note that even though Mood was a "better" trait than size at the outset, Size *given* Position is better than Mood given Position. However, if the expert could not provide information on Size, we would have to choose Mood as the next best alternative.

Each time a question is asked, new values for expected information are computed. Our goal is not to create a static decision tree, but simply to order questions dynamically in terms of their expected costs and benefits.

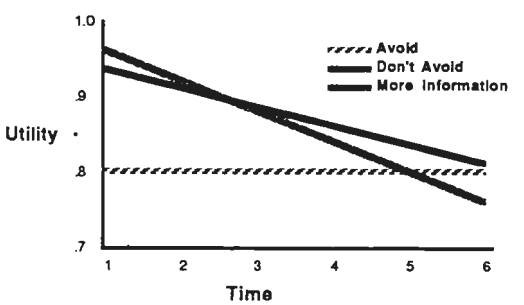


Figure 16: Expected value of the three alternatives over time.

Figure 16 shows the expected value of the three alternatives over time. If the insect hasn't flown off after 3 seconds, *Aquinas* would ask a question about its position. If it has come from the east, you can be certain it can't be a stinger, so you can rest easy. If it

has come from the west, the new information is incorporated and the preference model is re-evaluated to give the following plot:

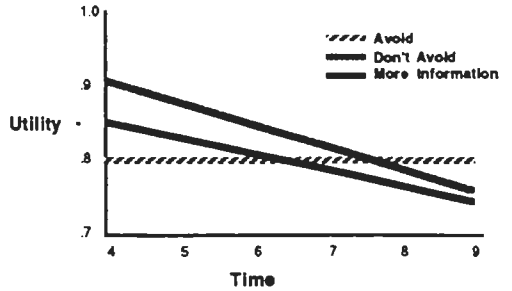


Figure 17: Expected value of the three alternatives given that you know the insect came from the West.

From figure 17, we conclude that if the insect is still around after 8 seconds, the best strategy is to avoid it.

SUMMARY

Personal Construct methodology and Decision Analytic techniques can be effectively combined to model complex problems involving informational and preferential components.

ACKNOWLEDGEMENTS

Thanks to Roger Beeman, Miroslav Benda, Kathleen Bradshaw, Stan Covington, Brian Gaines, Julia Jennings, Cathy Kitto, Sandra Marcus, Art Nagai, Doug Schuler, Mildred Shaw, David Shema, Lisle Tinglof-Boose, and Bruce Wilson for their contributions and support. Aquinas was developed at the Knowledge Systems Laboratory, Advanced Technology Center, Boeing Computer Services in Seattle, Washington. This paper was produced on an Apple® Macintosh™ Plus computer using Microsoft® Word™ and MacDraw™.

REFERENCES

Boose, J. H. (1984). Personal construct theory and the transfer of human expertise. *Proceedings of the National Conference on Artificial Intelligence*, Austin, Texas.

Boose, J. H. (1985). A knowledge acquisition program for expert systems based on Personal Construct Psychology. *International Journal of Man-Machine Studies*, **23**, 495-525.

Boose, J. H. (1986a). *Expertise Transfer for Expert System Design*. New York: Elsevier.

Boose, J. H. (1986b). Rapid acquisition and combination of knowledge from multiple experts in the same domain. *Future Computing Systems Journal*, 1, 191-216.

Boose, J. H. (1987). Uses of repertory grid-centered knowledge acquisition tools for knowledge-based systems. *Proceedings of the 2nd AAAI Knowledge Acuqisition for Knowledge-Based Systems Workshop*, Banff, Canada, October, 1987.

Boose, J. H. & Bradshaw, J. M. (1987a). Expertise transfer and complex problems: Using *Aquinas* as a knowledge-acquisition workbench for knowledge-based systems. *International Journal of Man-Machine Studies* **26**, 3-28.

Boose, J. H. & Bradshaw, J. M. (1987b). AQUINAS: A knowledge acquisition workbench for building knowledge-based systems. *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading, England, September, 1987.

Boulton, D.M. & Wallace, C.S. (1973). An information measure for hierarchic classification. *Computer Journal*, **16**, 254-261.

Cheeseman, P. (1983). A method of computing generalized Bayesian probability values for expert systems. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, August 1983.

Cheeseman, P. (1984). Learning of expert systems from data. *Proceedings of the Workshop on Principles of Knwledge-based Systems*, Denver, Colorado, December 1984.

Cheeseman, P. (1985). In defense of probability. *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, Los Angeles, August 1985.

Cheeseman, P. (1986). Probabilistic versus fuzzy reasoning. In L.N. Kanal and J. F. Lemmer (Eds.), *Uncertainty in Artificial Intelligence*, Amsterdam: Elsevier.

Cheeseman, P. (1987). Bayesian prediction for artificial intelligence. *Proceedings of the Third AAAI Workshop on Uncertainty in Artificial Intelligence*, Seattle, Washington, July 10-12, 1987.

Clancey, W. J. (1983). The epistemology of a rule-based expert system — A framework for explanation. *Artificial Intelligence*, **20**, 215-251.

Clancey, W. J. (1984). Classification problem solving. *Proceedings of the National Conference on Artificial Intelligence*, Austin, Texas.

Clancey, W. J. (1986). Heuristic Classification, in J. Kowalik (Ed.), *Knowledge-Based Problem-Solving*, New York: Prentice-Hall.

Diederich, J., Ruhmann, I. & May, M. (1987). KRITON: A knowledge acquisition tool for expert systems. *International Journal of Man-Machine Studies*, **26**(1).

Diederich, J., Linster, M., Ruhmann, I. & Uthmann, T. (1987). A methodology for integrating knowledge acquisition techniques. *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading, England, September, 1987.

Duda, R. O. & Reboh, R. (1984). AI and decision making: The PROSPECTOR experience. In W. Reitman (Ed.) *Artificial Intelligence Applications for Business*, Norwood, NJ.: Ablex.

Eco, U. (1986). In praise of Thomas Aquinas. *Wilson Quarterly*, Fall 1986.

Eshelman, L., Ehret, D., McDermott, J. & Tan, M. (1987). MOLE: A tenacious knowledge acquisition tool. *International Journal of Man-Machine Studies*, **26**, 41-54.

Gaines, B. R. (1987a). An overview of knowledge acquisition and transfer. *International Journal of Man-Machine Studies*, **26**, 453-472.

Gaines, B. R. (1987b) How do experts acquire expertise? *Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-based Systems*, Reading, England.

Gaines, B. R. & Shaw, M. L. G. (1981). New directions in the analysis and interactive elicitation of personal construct systems. In Shaw, M. L. G., Ed., *Recent Advances in Personal Construct Technology*. New York: Academic Press.

Gaines, B. R. & Shaw, M. L. G. (1986). Interactive elicitation of knowledge from experts. *Future Computing Systems*, **1**(2).

Garg-Janardan, C. & Salvendy, G. A conceptual framework for knowledge elicitation. *International Journal of Man-Machine Studies*, **27**(1).

Hayek (1952).

Heckerman, D.E. & Horvitz, E.J. (1987). On the expressiveness of rule-based systems for reasoning with uncertainty.

Henrion, M. (1987). Practical issues in constructing a Bayes' belief network. *Proceedings of the AAAI Workshop on Uncertainty*, Seattle, Washington.

Henrion, M. & Cooley, D. R. (1987). An experimental comparison of knowledge engineering for expert systems and decision analysis. *Proceedings of the National Conference on Artificial Intelligence*, Seattle, Washington.

Hink, R.F. & Woods, D.L. (1987). How humans process uncertain knowledge: An introduction for knowledge engineers. *AI Magazine*, Fall 1987, 41-53.

Hinkle, D. N. (1970). The game of personal constructs. In D. Bannister (Ed.) *Perspectives in Personal Construct Theory*. London: Academic Press.

Holtzman, S. H. & Breese, J. (1986). Exact reasoning about uncertainty: On the design of expert systems for decision support. In L.N. Kanal and J. F. Lemmer (Eds.), *Uncertainty in Artificial Intelligence*, Amsterdam: Elsevier.

Holtzman, S. H. (1988). *Intelligent Decision Systems*. Reading, MA: Addison-Wesley. In press.

Howard, R. A. & Matheson, J. E. Eds. (1984). *Readings on the Principles and Applications of Decision Analysis*, Menlo Park, California: Strategic Decisions Group.

Keeney, R. L. (1986). Value-driven expert systems for decision support. Los Angeles: Systems Science Department, University of Southern California.

Keeney, R. L. & Raiffa, H. (1976). *Decisions with multiple objectives: Preferences and value tradeoffs*. New York: Wiley.

Kelly, G. A. (1955). *The Psychology of Personal Constructs*. New York: Norton.

Kim, J. & Pearl, J. (1983). A computational model for combined causal and diagnostic reasoning in inference systems. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, August 1983.

King, A. H. (1979). Mechanization.

Kitto, C. & Boose, J. H. (1987). Heuristics for expertise transfer: The automatic management of

complex knowledge acquisition dialogs. *International Journal of Man-Machine Studies*, **26**, 183-202.

Langlotz, C. P., Shortliffe, E. H. & Fagan, L. M. (1986). Using decision theory to justify heuristics. *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, Pennsylvania.

Marcus, S. (1987). Taking backtracking with a grain of SALT. *International Journal of Man Machine Studies*, **26**, 383-398.

Pearl, J. (1985). How to do with probabilities what people say you can't. *Proceedings of the Second Annual Conference on Artificial Intelligence Applications*, Miami, Florida, December 11-13, 1985.

Pearl, J. (1986). Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, **29** 241-288.

Pribram, K. (1971).

Quinlan, J.R. (1983). Learning efficient classification procedures and their application to chess endgames. In R.S. Michalski, J.G. Carbonell, & T.M. Mitchell (Eds.) *Machine Learning: An Artificial Intelligence Approach*, Palo Alto, CA: Tioga.

Quinlan, J. R. (1986). The effect of noise on concept learning. In R.S. Michalski, J.G. Carbonell, & T.M. Mitchell (Eds.) *Machine Learning: Volume 2*, Morgan Kaufman.

Raiffa, H. (1968). Decision analysis: Introductory lectures on choices under uncertainty. Reading, Mass.: Addison-Wesley.

Saaty, T. L. (1980). *The Analytic Hierarchy Process*. New York: McGraw-Hill.

Shastri, L. (1985). *Evidential Reasoning in Semantic Networks: A Formal Theory and its Parallel Implementation*. Technical Report 166, Rochester, NY: University of Rochester, September 1985.

Shastri, L. (1987). A connectionist encoding of semantic networks. *Proceedings of the Cognitive Science Society*, Seattle, Washington.

Shastri, L. & Feldman, J. (1985). Evidential reasoning in semantic networks: A formal theory. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, California.

Shaw, M. L. G. & Gaines, B. R. (1987). Techniques for knowledge acquisition and transfer. *International Journal of Man-Machine Studies*. In press.

Spiegelhalter, D. J. (1986a). A statistical view of uncertainty in expert systems. In W. Gale, Ed., *AI and Statistics*. Reading, MA: Addison-Wesley.

Spiegelhalter, D. J. (1986b). Probabilistic reasoning in predictive expert systems. In L.N. Kanal and J. F. Lemmer (Eds.), *Uncertainty in Artificial Intelligence*, Amsterdam: Elsevier.

Weimer, W. (1975).

Wellman, M. P. (1986). Reasoning about assumptions underlying mathematical models. In J. S. Kowalik (Ed.), *Coupling Symbolic and Numeric Computing*. Amsterdam: Elsevier.

Winograd, T. & Flores, F. (1987). *Understanding Computers and Cognition*. Reading, Mass.: Addison-Wesley.

Wise, B. P. & Henrion, M. (1986). A framework for comparing uncertain inference systems to probability. In L.N. Kanal and J. F. Lemmer (Eds.), *Uncertainty in Artificial Intelligence*, Amsterdam: Elsevier.

---

[1] *Aquinas* is named for the twelfth century theologian and scholar, St. Thomas Aquinas:

Aquinas christianized Aristotle, thus reconciling faith and reason... [Before St. Thomas,] when the text of an ancient author was studied, the commentator or copyist, when he came upon something that clashed with revealed religion, either scratched out the 'erroneous' sentences, or else shifted the words to the margin. What did Thomas do instead? He aligned the divergent opinions, clarified the meaning of each, questioned everything, even the revealed datum, enumerated the possible objections, and essayed a final mediation. (Eco, 1986)

[2] We have followed Holtzman (1988) for many of the elements of this discussion, although some of the details, content, and organization of the discussion differ.

[3] This modeling is not restricted to the symbolic activity of 'conscious' thought, but also includes such things as assimilation, accommodation, and habituation — indeed, "virtually all neural activity is symbolic and... words and conscious awareness are inessential to this mode of functioning":

> "If symbolism is defined in a very general way, as the ability of processes to parallel or imitate each other, then it becomes clear that other aspects of neural activity are equally as symbolic as thought. Indeed, *modeling is the fundamental function of the central nervous system...*" (Weimer, 1975, p. 459)

Karl Pribram (1971) has shown that habituation does not indicate a loss of sensitivity by the nervous system; rather, the brain develops a neural model of the environment, a representation, an expectancy against which inputs are constantly matched. But this is only half the story: "[T]o this picture must be added the startling rider that the nervous system *creates* its own inputs" (Weimer, 1975, p. 460).

[4] King continues: "This is my answer to all people who are shallow enough to believe in optimism." (King, 1979).

[5]Different approaches have been taken to the problem of commensuration (measuring distinct attributes together) of multiple objectives and the measurement of risk attitude. (Keeney & Raiffa, 1976) derive a multiattribute utility function by modeling attribute commensuration and risk attitude simultaneously, whereas (Howard & Matheson, 1984) favor the separation of the commensuration process from risk-attitude modeling.

[6] A value is significant, in an information theoretic interpretation, when the minimum message length required to encode it, given its expected value, is longer than the minimum message length to encode the observed probability ignoring its expected value.

[7] Since the search procedure becomes combinatorially explosive for higher order joint probabilities, it is suggested that they only be investigated when they have at least one significant lower order (i.e., binary) component joint probability.

[8] This is simply a multi-dimensional contingency table in a slightly altered format.

[9] Unfortuately, it is difficult to represent events that only occur once in a contingency table.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

What Kind of Information Processing is Intelligence? A Perspective on AI Paradigms and A Proposal [1]

B. Chandrasekaran
Laboratory for AI Research
The Ohio State University
Department of Computer & Information Science
Columbus, OH 43210

Summary

The only truly universally shared paradigm among AI workers (and its friends in other disciplines) is: "Significant aspects of cognition and perception are usefully understood as *information processing activities on representations.*" While this characterization rules out needing to worry about chemistry and holograms for capturing the essence of information processing (IP) in intelligence, it does not characterize the nature of intelligent IP sufficiently within the class of all information processing activities. What is the *essential* nature of intelligence as information processing that can be used to characterize all its manifestations: human, alpha-centaurian and artificial?

In this talk we will trace the historical currents in AI with a view to giving a sense for various answers to this question, and concentrate, for convenience, on cognition , though we think appropriate extensions can be made for perception. Even within representational theories, there are two streams. The so-called symbolic paradigm, the traditional one in AI, works on the hypothesis that the representations are discrete symbol systems, which are processed by algorithmic interpretations. More recently, the connectionist school has suggested that the representations are non-symbolic, and proposed an alternative representation in the forms of weights of connections in a network.

I will discuss the connectionist-symbolic debate briefly, but concentrate

---

[1]Research supported by DARPA/RADC Contract F30602–85–C–0010, NSF grant MCS–8305032 and AFOSR grant 82–0255. Earlier versions of this talk have been presented at the AAAI Workshops on Foundations of Artificial Intelligence, February 6–7, 1986, Las Cruces, NM, and on Theoretical Issues in Conceptual Information Processing, Philadelphia, PA, 1986, and Washington, D. C., 1987.

mostly on the major symbolic paradigms. Most of the early work in AI within the symbolic IP framework merely served to show that many mental functions were *computable*, and created, in the form of programs, "realities in the field" that performed versions of tasks that in humans clearly require intelligence. The net effect of this was to make the idea of AI more plausible. However, with additional work in the following decade, three broad kinds of theories about intelligence constituting different answers to the question about the nature of underlying IP:

1. Architectural theories, which sought or proposed a level of abstraction and a corresponding IP architecture at which intelligence *qua* intelligence emerges. Above this level are lots of particulars of an agent, and below is implementation.

2. Logical theories which describe an intelligent agent abstractly as a knower of propositions (a significant part of these propositions would deal with common sense knowledge of the world) and which typically propose the use of *deductive inference* as the IP mechanism to produce intelligence. Actual intelligent agents that exist, e.g., human beings, capture and implement this abstract design more or less well by using *heuristics.*

3. Functional theories of AI that embody one or more generic, functional properties of intelligence to solve a "natural kind" of a cognitive problem. These theories typically emphasize generic *organizational* aspects, which facilitate particular *classes of inference* in a *computationally efficient* way.

We will briefly outline our reasons for believing that types 1 and 2 above do not provide a satisfactory answer to the question about the nature of intelligence, and cast our sympathies with theories of type 3. We will point out that GPS, frames, some of Schank's theories and our own theories of knowledge-based problem solving are examples of type 3 theories. We will use *classification* as an example of an IP strategy to both illustrate a theory of this type and as a means to introduce and motivate our proposal about the essential nature of the IP operations that constitute intelligence:

> The basic constraint to which intelligence as IP is a response is one of *computational complexity* of the input/output problem faced by the agent. Intelligence is a coherent repertoire of generic information-processing strategies, each of which solves an information processing problem in a computationally efficient way, using knowledge of a certain type, organized in a particular way, and using a particular control regime.

Strategies of this type are what unite Einstein and the "man on the street" (and alpha-centaurians). While theories of type III do not by themselves

constitute psychological theories, they nevertheless *provide the formalisms for couching cognitive theories.* They characterize intelligence abstractly to the extent that they define IP tasks, and propose knowledge forms and processing mechanisms to match the needs of the task. It also seems to us that the union of knowledge representation primitives for these tasks together constitute a good part of *mentalese.* Finally, the proposed view is consistent with the multiplicity of structures and mechanisms that one would expect of *intelligence as a biological phenomenon*, evolved to solve real problems of agents in real time.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

Extensions to the CART Algorithm

Stuart L. Crawford*
Advanced Decision Systems
201 San Antonio Circle
Mountain View, CA 94040
stuart@ads.arpa

Abstract

The CART concept induction algorithm recursively partitions the measurement space, displaying the resulting partitions as decision trees. Care, however, must be taken not to overfit the trees to the data, and CART employs cross-validation (*cv*) as the means by which an appropriately sized tree is selected. Although unbiased, *cv* estimates exhibit high variance, a troublesome characteristic, particularly for small learning sets. This paper describes Monte Carlo experiments which illustrate the effectiveness of the *.632 bootstrap* as an alternative technique for tree selection and error estimation. In addition, a new *incremental learning* extension to CART is described.

1 INTRODUCTION

It is widely acknowledged that knowledge aquisition represents a major knowledge engineering bottleneck (Feigenbaum, 1981). Algorithms that induce rules from a collection of examples automatically construct knowledge bases, partially alleviating this bottleneck. This paper focuses upon the CART (*Classification and Regression Trees*) algorithm (Breiman, Friedman, Olshen & Stone, 1984) as a tool for concept induction and, in particular, describes two important extensions to the CART approach: the introduction of an *incremental learning* capability, and the definition of an improved *error estimation* methodology.

*Supported by Advanced Decision Systems IR&D Funds.

The development of algorithms for concept induction has a rich history, beginning with Hunt's *Concept Learning System* (Hunt, Marin & Stone, 1966), and followed by Quinlan's *ID3* algorithm (Quinlan, 1979). During that time interval the statistical community was, in parallel, developing powerful procedures for classification based upon recursive partitioning algorithms[1]. It is the premise of this paper that the most powerful of these recursive partitioning based classifiers is, in fact, a remarkably sophisticated tool for concept induction.

The early induction schemes of Hunt and Quinlan, however, were restrictive, as they assumed that the class distributions did not overlap in the measurement space and that the collection of examples was measured without error. In other words, it was presumed that the domain and learning set were such that *perfect classification* was possible. These algorithms also imposed the restriction that examples be composed only of attributes whose values ranged over a small unordered set of discrete values and were, in addition, unable to make use of examples for which some attribute values were missing. Because these restrictions and assumptions are clearly unrealistic in many situations, early concept induction algorithms were criticized as being useful only in constrained domains.

As research in concept induction proceeded, many of these restrictions were relaxed. The *ACLS* procedure (Patterson, 1983) permitted integer valued attributes, and *ASSISTANT* (Kononenko, Bratko & Roskar, 1984) allowed

[1] See Crawford (1987) for a review.

real valued attributes, as well as learning sets that could be both noisy and incomplete. Allowing learning sets to be composed of attributes measured with error and drawn from overlapping class distributions is in itself problematic, since the induced decision tree will almost certainly *overfit* the learning set. Although statisticians often talk about the problem of overfitting in terms of a tradeoff between bias and variance, in the context of concept induction, the crux of the problem is essentially that the induced tree is unnecessarily complex, making it both less interpretable and more error prone than a tree of the "right size".

Given the problems imposed by a noisy domain, much of the recent concept induction literature has focused upon the difficulty of deriving efficient node splitting criteria and reasonable rules for stopping tree growth. Unfortunately, these derivations have seldom been approached with much statistical rigor. A notable exception to these shortcomings is the research of Breiman *et al* (1984), in which the CART algorithm is described. CART takes as input a learning set, $\mathcal{L}$, whose measurement vectors may be composed of attributes whose values are boolean, integer, real or nominal. In addition, the data may be both noisy and sparse. The researcher has the option of specifying both a loss function and prior probabilities of class membership, parameters which can be of crucial significance, particularly in diagnostic situations[2]. The output from CART is a disjunctive set of decision rules, represented in the form of a binary decision tree. The tree is a parsimonious, easily interpretable representation of induced domain expertise with which new observations can be quickly classified. In addition, trees of maximal size generated from CART have an error rate bounded by twice the Bayes error rate, and can be shown to be asymptotically Bayes optimal (Gordon & Olshen, 1978).

In the sections to follow, the newly developed error estimation and incremental learning extensions to CART are described.

2 ESTIMATING ERROR

2.1 Introduction

In this section, we begin by briefly outlining the CART algorithm itself. This is followed by descriptions of a new approach to estimating the error rate of trees induced by CART, as well as results of experiments designed to evaluate the performance of this new approach.

2.1.1 The CART Algorithm

Overview This section provides a brief[3] overview of the CART methodology. For simplicity, it is assumed throughout that class prior probabilities are computed as sample proportions and that a simple 0-1 loss function is specified. Note, however, that simple modifications to expressions 2 and 4 allows for the accomodation of asymmetric costs and user specified prior probabilities.

The CART algorithm proceeds as follows: A classification tree, T, initially consists only of the root node τ_1, containing all of the cases in $\mathcal{L}$[4]. A search is made through the set of possible binary splits of τ_1 until an optimal split s_{opt} is found. For real-valued attributes A, splits take on the form $A \leq r$, where r is a midpoint of any adjacent pair found in the set of unique values attained by A. For categorical attributes which can attain possible values $D_1, \ldots, D_M$, splits take on the form $A \in \{D\}$ where $\{D\}$ is one of the subsets of $\{D_1, \ldots, D_M\}$. Once s_{opt} is located, the root node acquires two descendants, τ_L and τ_R, each of which now contains a subset of $\mathcal{L}$. Ge-

[2] Consider a situation in which $\mathcal{L}$ consists of cases for which the class assignment is either *diseased* or *healthy*. Suppose further that $\mathcal{L}$ contains equal numbers of each class. If the prior probability of disease is known to be very small, then an induction procedure which fails to take priors into account will produce a rule with a bias towards the misclassification of healthy patients as diseased. This difficulty is exacerbated if the *cost* of such a misclassification is higher than the cost of misclassifying diseased patients as healthy.

[3] For a more detailed treatment refer to Therneau (1983), for a complete analysis of alternative procedures for tree error rate estimation, refer to Crawford (1987) and for the definitive description of the methodology, refer to Breiman *et al* (1984).

[4] $\mathcal{L}$ consists of N cases each of which consists of a *known* class θ and a vector of attributes.

ometrically, s_{opt} describes a partition orthogonal to one of the coordinates of the measurement space. The partitioning process is applied recursively to each leaf, continuing until all leaves are too small, contain cases from a single class only, or contain identical measurement vectors. When partitioning ends, the initial tree has grown to become a terminal tree T_{max}. Figure 1 shows how a two dimensional measurement space can be partitioned. The scatterplot on the left illustrates the distribution of two classes (represented by solid and hollow dots) in the coordinate plane. Lines perpendicular to the coordinate axes, and labelled a through d, illustrate how the space could be partitioned in order to correctly classify the observed cases. The binary tree on the right illustrates how such a partitioning can be represented as a decision rule. Each interior node has a label corresponding to the appropriate partition, and asks a question of the form "is $x > c$?". Arcs represent binary responses to the questions and leaves represent class assignments[5].

Since, with a noisy domain, T_{max} almost certainly overfits $\mathcal{L}$, it is pruned backwards, yielding a unique, nested sequence of subtrees $\{T_{max} \preceq, \ldots, \preceq \tau_2 \preceq \tau_1\}$, and an error estimation procedure is then applied to the sequence in order to select an optimally sized subtree. Subsequent observations are dropped down the selected tree, and classified according to the class membership of the plurality of cases populating the leaf into which they fall.

Splitting Criteria Since the tree building mechanism attempts to generate a tree with increasingly homogeneous (pure) nodes, a natural splitting criterion is one which tends to maximize some index of purity. For C classes, using an impurity function ϕ, and defining $p(j|\tau)$ as the proportion of class j cases at node τ, we define the impurity of node τ as: $\iota(\tau) = \phi(p(1|\tau), \ldots, p(C|\tau))$. Any split s on τ sends a proportion p_L of cases to τ_L and a proportion p_R to τ_R, and the decrease in impurity obtained from splitting τ by s is defined by

$$\Delta\iota(s,\tau) = \iota(\tau) - p_R\iota(\tau_R) - p_L\iota(\tau_L) \quad (1)$$

It turns out that the performance of the classification tree algorithm is relatively insensitive to the choice of ϕ, as long as certain concavity requirements are met. Some interesting candidates for ϕ are the information index (as used in *ID3*) and the Gini[6] index of diversity, defined as:

$$\sum_{i=1}^{C}\sum_{j=1}^{C} p(j|\tau)p(i|\tau), i \neq j. \quad (2)$$

When faced with evaluating categorical splits of the form $A \in \{D\}$, many other binary tree based concept induction algorithms have been thwarted by combinatorial explosion since, when locating the best split for an M-valued categorical attribute, these algorithms must examine $2^{M-1} - 1$ subsets. However, using a generalization of a result due to Fisher (1958), CART dramatically reduces the search space required for the location of optimal splits for categorical variables. When $C = 2$ and the attribute values are ordered such that

$$p(1|A = D_{(1)}) \leq, \ldots, \leq p(1|A = D_{(M)}) \quad (3)$$

an optimal split may be obtained via a reduced search through one of the M subsets $D_{(1)}, \ldots, D_{(h)}; h = 1, \ldots, M$. When $C > 2$, the C known classes can be explicitly partitioned into two disjoint sets (C_1, C_2) and, for each partition, search reduced via 3. The optimal split for each partition is stored and the best of these splits is returned as optimal. When explicit class-partitioning is employed via 3 a search of size $M(2^{C-1} - 1)$ is necessary, resulting in dramatic savings for small C and large M.

Pruning the Tree As described previously, CART avoids the pitfalls of stopping rules, by first growing a tree of maximal size, and then pruning backwards in an appropriate manner.

[5] It is worth noting that, with no additional cost, CART can deliver estimates of *probabilities* of class membership at the leaves.

[6] Corrado Gini, 1884-1965

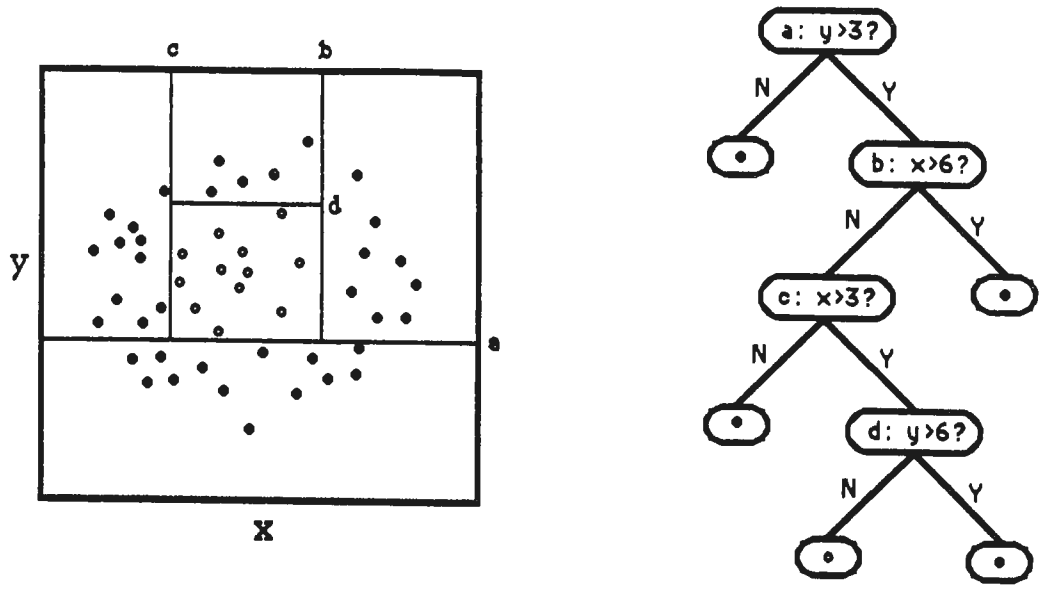


Figure 1: A Simple Partitioning

The pruning approach is based upon the notion of assigning a complexity penalty to the growth of large trees. Defining the cost complexity of tree T as $R_\alpha(T) = \hat{R}(T) + \alpha T$, where α is a real-valued scalar, T is the number of leaves in T and $\hat{R}(T)$ is the *resubstitution estimate*[7] of $R^*(T)$ (the *true* error rate of T) we can, without exhaustive search, locate that ***unique*** smallest subtree $T(\alpha)$ of T_{max} which minimizes $R_\alpha(T)$ for fixed α. Although it is not implemented in this manner, minimal cost-complexity pruning can be thought of as a procedure in which α is steadily increased, with pruning occurring at threshold values α', thus yielding a nested sequence of minimizing subtrees.

Error Estimation Once the sequence of minimizing subtrees has been obtained, a minimum risk subtree, T_{opt} is sought. If $\hat{R}(T)$ is the criterion used to select T_{opt} then T_{max} will always be selected, since $\hat{R}(T)$ decreases with increasing T. However, since $\hat{R}(T)$ is obtained as a result of resubstituting $\mathcal{L}$ into a tree induced from $\mathcal{L}$, $\hat{R}(T)$ is a *downwardly biased* estimate of $R^*(T)$ and therefore inappropriate.

A *cross-validation* (cv) technique for selecting an unbiased estimate of $R^*(T)$ randomly divides $\mathcal{L}$ into V test samples, $\mathcal{L}_1, \ldots \mathcal{L}_V$, each of which has a (nearly) identical number of cases, and V learning samples $\mathcal{L}_{(v)} = \mathcal{L} - \mathcal{L}_v$ are created. For each $\mathcal{L}_{(v)}$, an auxiliary tree, $T_{(v)}$ of maximimum size is grown and pruned according the the cost-complexity values established when T_{max} was originally pruned. Since the set of subtrees established from each $\mathcal{L}_{(v)}$ was generated without knowledge of the cases in $\mathcal{L}_v$, the cases in $\mathcal{L}_v$ can serve as an *independent test sample* for those subtrees. Now, for fixed α, $\mathcal{L}_v$ is dropped down the corresponding $T_{(v)}$ each of which yields $N_{ij(v)}$, the number of class j cases in $\mathcal{L}_v$ classified as belonging to class i. Summing across all V, the number N_{ij} of misclassified class j cases is obtained for all i,j. Since the total number N_j of class j cases in *all* test samples is identical to the total number of class j cases in $\mathcal{L}$ then the probability that a class j case is misclassified is estimated by $\frac{N_{ij}}{N_j}$ and a *cross-validation* estimate of misclassification rate is obtained by:

$$\hat{R}^{cv}(T_\alpha) = \sum_{i=1}^{C} \sum_{j=1}^{C} \frac{N_{ij}}{N_j} \qquad (4)$$

Finally, 4 is calculated for each α, α_{opt} is selected as that α for which 4 is a minimum, and

[7] The estimate obtained by classifying the cases in $\mathcal{L}$ by the *same* tree used to induce $\mathcal{L}$.

T_{opt} is selected from the sequence of minimizing subtrees grown from T_{max} as that subtree corresponding to the optimal complexity value α_{opt}.

2.2 Alternatives to *cv*

2.2.1 Introduction

Since $\hat{R}(T)$ underestimates $R^*(T)$ we define the random variable *op*[8] as $op = R^*(T) - \hat{R}(T)$ with expectation ω. If ω was observable, we could estimate $R^*(T)$ by $\hat{R}(T) + \omega$ but since, of course, ω is unobservable, it must be estimated from $\mathcal{L}$. Recalling $\hat{R}^{cv}(T)$, the *cv* estimate of $R^*(T)$, we can estimate ω by $\hat{\omega}^{cv} = \hat{R}^{cv}(T) - \hat{R}(T)$.

The Bootstrap An alternative estimate of ω may be obtained via the *bootstrap*[9] resampling procedure (Efron, 1982) in which $\hat{\omega}^{boot}$ estimates $\omega(f)$, where f is the empirical distribution obtained by placing mass $\frac{1}{N}$ on each case from $\mathcal{L}$. The bootstrap procedure is carried out as follows: B independent learning samples $\mathcal{L}^{*1}, \mathcal{L}^{*2}, \ldots, \mathcal{L}^{*B}$ each of size N, are created by random sampling (with replacement) from $\mathcal{L}$ and, from each, an auxiliary tree of maximal size is grown. Error is subsequently estimated by classifying the cases in $\mathcal{L}$ by the auxiliary tree grown from $\mathcal{L}^{*b}$. Defining P_n^{*b} as the resampled proportion of each case x_n in $\mathcal{L}^{*b}$, then a bootstrap replication of *op* is obtained from each auxiliary tree as:

$$op^{*b} = \sum_{n=1}^{N} (\frac{1}{N} - P_n^{*b}) I_{(\theta, \hat{\theta})} \qquad (5)$$

where I is an indicator function evaluating to unity when the true class, θ, differs from the assigned class $\hat{\theta}$. Examination of 5 reveals that cases from $\mathcal{L}$ that are absent from $\mathcal{L}^{*b}$ contribute a positive increment to op^{*b} when subsequently misclassified by the auxiliary tree. Averaging 5 over all B bootstrap samples yields $\hat{\omega}^{boot}$ from which we obtain $\hat{R}^{boot}(T) = \hat{R}(T) + \hat{\omega}^{boot}$. In a manner similar to that of the *cv* approach described above, by pruning all B auxiliary trees according to the previously computed values of α and evaluating 5 one can select T_{opt} as that tree in the original nested set which corresponds to α_{opt}.

In a series of Monte Carlo trials, Efron (1983) evaluated both $\hat{\omega}^{boot}$ and $\hat{\omega}^{cv}$ for a linear discriminant analysis procedure in which *op* could be calculated for each trial. He showed that although $\hat{\omega}^{cv}$ more closely approximated ω than did $\hat{\omega}^{boot}$, which was biased downwards, $\hat{\omega}^{boot}$ was much less variable than $\hat{\omega}^{cv}$. Efron also illustrated that $\hat{\omega}^{.632}$, a new estimator, performed best of all, exhibiting both low variance *and* low bias. The research described in this section was designed to ascertain whether $\hat{\omega}^{.632}$ could also prove to be a useful estimator in the context of CART.

The .632 Bootstrap Geometrically, $\hat{R}(T)$ is the error rate for new cases zero distance from $\mathcal{L}$, whereas $R^*(T)$ is the expected error rate for new cases which *may* fall some distance from $\mathcal{L}$. For the ordinary bootstrap, there is a high probability ($1 - e^{-1} \cong .632$) that a case in $\mathcal{L}$ will fall zero distance from a case in $\mathcal{L}^{*b}$. If $\varepsilon^{(o)}$ is defined as the error rate for those cases in $\mathcal{L}$ which are *absent* from $\mathcal{L}^{*b}$, then we find that the cases contributing to $\varepsilon^{(o)}$ are too far away (about $\frac{1}{.632}$ too far out on the distance axis), whereas the remaining points are too close (zero distance). The weighted average

$$\hat{R}^{.632}(T) = .368\hat{R}(T) + .632\varepsilon^{(o)}$$

yields an estimate of misclassification rate for cases about the right expected distance from $\mathcal{L}$, and should therefore be roughly unbiased.

2.2.2 Experiment I

Methodology Assume an *oracle* which can infallibly select T_{opt} from the nested sequence of minimizing subtrees. Which error estimation method, under a variety of different probability scenarios and sample sizes, is most likely to provide the best estimate of the true error rate of T_{opt} ? Experiment I is designed to answer this question, and proceeds as follows:

[8] Short for *optimism*.

[9] Henceforth called the *ordinary bootstrap*.

(i) For each combination of probability model and sample size, do (ii), (iii).

(ii) Draw $\mathcal{L}$ of sample size n from model M, build T_{max} and prune. Using an independent test sample[10], determine T_{opt} and compute estimates of $R^*(T)$ via *cv*, *ordinary bootstrap*, and *.632 bootstrap* techniques. Repeat for 100 trials.

(iii) Assess and compare the performance of each estimation technique.

The performance of an estimator is assessed by measuring how well, *on average*, $\hat{R}^*(T)$ estimates $R^*(T)$ for a given $\mathcal{L}$. An appropriate measure is the *mean squared error* (*MSE*), calculated as:

$$\begin{aligned} MSE &= E(\hat{R}^*(T) - R^*(T))^2 \qquad (6) \\ &= E(\hat{\omega} - op)^2 \\ &= E(\hat{\omega} - \omega)^2 + \sigma^2_{\hat{\omega}} - 2\sigma_{\hat{\omega},op} + \sigma^2_{op} \end{aligned}$$

The performance of each estimator can be conveniently compared by comparing the *relative inefficiency* (*REL*) of the mean squared errors according to the following scale:

$$REL = \frac{MSE - MSE^{ic}}{MSE^{zero} - MSE^{ic}}$$

where MSE^{zero} is the *MSE* associated with $\hat{R}(T)$ and MSE^{ic} is associated with the *"ideal constant"* estimator, $\hat{R}^{ic}(T) = \hat{R}(T) + \omega$. The estimator of risk associated with the lowest *REL* is that estimator whose performance is expected to most closely approach $R^*(T)$.

The Learning Sets The learning sets used for Experiment I were drawn from data generated by five different probability models, as follows:

A1 Two classes represented by *Gaussian* spheres in $\Re^4$, and separated by a Euclidean distance of 2.

[10]Independent test samples of size 5000 were used throughout.

	cv	*boot*	*.632*
A1	5.61	16.83	3.24
A2	6.84	15.12	2.31
B	7.54	4.23	-5.22
C	6.82	22.79	3.43
D	1.21	23.40	2.01

Table 1: *REL, N=100*

A2 Identical to A1, save for the addition of two *pure noise* attributes, both of which are drawn from a *Gaussian (0,1)* distribution.

B Identical to A1, except that the distribution is *Cauchy*.

C One class is drawn from a uniformly distributed, four dimensional hypersphere with inner radius 3 and outer radius 4, and the other class is drawn from a spherically *Gaussian (0,1)* distribution. Both classes are located at the origin, so one class is simply nested within the hypersphere formed by the other.

D The two classes are drawn from a *Gaussian* distribution located at the origin, and differ only in covariance structure.

Details of the probability models, as well as the programs used to generate them, may be found in the appendix.

Results Table 1 summarizes the main results of Experiment I for $\mathcal{L}$ of size 100, and Table 2 summarizes the results when $\mathcal{L}$ has only 20 cases. Recall that a *REL* of 100% indicates that $\hat{R}^*(T)$ estimates $R^*(T)$ no better than $\hat{R}(T)$ whereas a *REL* of 0% indicates that $\hat{R}^*(T)$ performs just as well as the "*ideal*" estimator $\hat{R}^{ic}(T)$.

Note that the *ordinary bootstrap* provides a poor estimate of $R^*(T)$ for trials of both sample sizes and that *cv* also performs poorly when $N = 20$. The performance of *cv* improves rather dramatically, however, when sample size is increased to 100 and, in fact, provides the best estimate for model D. The

	cv	*boot*	*.632*
A1	35.74	31.38	11.87
A2	28.97	28.96	8.48
B	44.77	22.16	1.51
C	25.57	31.49	8.02
D	13.08	22.94	-5.10

Table 2: *REL, N=20*

.632 bootstrap performed well for both sample sizes, and was particularly effective for small $\mathcal{L}$. Decompositions[11] of the *MSE* components into the terms shown in 6 illustrate that the poor performance of the *ordinary bootstrap* is largely due to inflated *bias* terms[12], whereas inflated *variance* terms[13] account for most of the error for *cv*. The *.632 bootstrap*, on the other hand, is characterized by moderate bias *and* variance terms.

2.2.3 Experiment II

Methodology The results of Experiment 1 indicate that, given T_{opt}, the *.632 bootstrap* is most likely to deliver the best estimates of $R^*(T)$, particularly when $\mathcal{L}$ is small. Recall, however, that CART relies upon an error estimation scheme as a means of *selecting* a tree in addition to estimating $R^*(T)$. It is therefore natural to ask which of the error estimation schemes is most likely to lead to the *selection* of T_{opt}. It is by no means certain that the *.632 bootstrap*, although efficient at error estimation *given* T_{opt}, will also prove to be the method of choice for selecting T_{opt}.

Because of computational expense, the scope of this second experiment is reduced to consider only *cv* and *.632 bootstrap* estimators. Experiment II proceeds as follows:

(i) For each combination of sample size and probability model do (ii), (iii).

(ii) Draw $\mathcal{L}$ of sample size n from model M, build T_{max} and prune. Using an independent test sample, determine T_{opt}. Using the *cv* and *.632* estimators select *hypothetical* minimum error trees (T_{sel}). Repeat for 100 trials.

(iii) Using the *MSE* criterion, assess and compare the performance of each selection technique.

As in Experiment I, some criterion against which to measure the performance of the two estimators is required. The *MSE* criterion of 6 cannot, however, be used as is. Whereas Experiment I involved comparing $R^*(T)$ for T_{opt} with $\hat{R}^*(T)$ for T_{opt}, Experiment II is rather more complicated, and involves comparing $R^*(T)$ for T_{opt} with $\hat{R}^*(T)$ for T_{sel}. Since T_{opt} and T_{sel} may differ, *MSE* is in fact $MSE(\hat{R}^*(T_{sel}))$ and is calculated as:

$$\begin{aligned} MSE &= E(R^*(T_{opt}) - \hat{R}^*(T_{sel}))^2 \qquad (7) \\ &= E(\hat{R}(T_{sel}) - \hat{R}(T_{opt}) + \hat{\omega}_{sel} - op_{opt})^2 \end{aligned}$$

Experiment I employed an index of *relative inefficiency* (*REL*) with which one could compare $MSE(\hat{\omega})$ relative to the "best case" situation MSE^{ic} in which $\hat{\omega} = \omega$, and the "worst case" situation MSE^{zero} in which $\hat{\omega} = 0$. In a similar manner, a *REL* index is derived for Experiment II, and 7 is compared relative to the two extremes as follows:

- When $\hat{\omega} = 0$, that tree for which $\hat{R}(T)$ is a minimum is selected, and hence $T_{sel} = T_{max}$. Now, $\hat{R}(T_{max}) = 0$,[14] and hence $\hat{R}^*(T_{sel}) = \hat{R}(T_{max}) + \hat{\omega} = 0$. Therefore, MSE^{zero} is simply $E(R^*(T_{opt}))^2$, the expected squared risk of the optimal tree.

- When $\hat{\omega} = \omega$, let $T_{sel} = T_{opt}$ and so

$$\begin{aligned} MSE^{ic} &= E(\hat{\omega} - op)^2 \\ &= E\{E(op) - op\}^2 \\ &= \sigma^2_{op} \end{aligned}$$

[11] Not shown here.
[12] $E(\hat{\omega} - \omega)^2$.
[13] $\sigma^2_{\hat{\omega}}$.

[14] In some situations, particularly when the *Bayes* error rate of the model is very high, leaves of T_{max} may contain cases with identical attribute vectors, despite the fact that they belong to different classes. In such situations, the class of the leaf is assigned arbitrarily and $\hat{R}(T_{max}) > 0$. Such situations did not, however, arise here.

	R^B	*cv*	*.632*
C	.207	6.25	1.53
A1	.248	2.08	1.46
A2	.259	2.05	1.48
D	.323	0.65	2.10
B	.329	1.29	1.19

Table 3: *REL, N=100*

	R^B	*cv*	*.632*
A1	.288	28.86	7.14
A2	.301	20.69	4.24
C	.355	16.48	4.66
B	.368	10.17	-0.95
D	.424	10.22	-5.08

Table 4: *REL, N=20*

Results Table 3 summarizes the main results of Experiment II for $\mathcal{L}$ of size 100, and Table 4 summarizes the results when $\mathcal{L}$ has only 20 cases. Note that these results are tabulated in order of *increasing* R^B, where $R^B$is the *Bayes* error rate of the probability model, as estimated from an independent test sample.

Table 3 suggests that, in terms of relative inefficiency, there is not a great deal of difference in the performance of the two estimation methods, although there is a suggestion that the performance of the *cv*, relative to the *.632 bootstrap*, may deteriorate for low error rate trees. When sample size is reduced to 20, however, the large variance of *cv* takes its toll, and the *.632 bootstrap* clearly outperforms *cv*.

2.2.4 Experiment III

Methodology The suggestion in Table 3 that, for large $\mathcal{L}$ at least, the performance of the estimators is dependent upon R^B is sufficiently interesting to warrant a third investigation, in which the range of R^B is more adequately sampled. Subsequent to the addition of seven auxiliary data sets (described below), Experiment II is repeated for sample sizes of 100 only.

	F1	*F2*	*F3*	*E1*	*E2*	*E3*	*E4*
δ_E	8.0	6.0	3.2	0.8	0.4	0.2	0.0

Table 5: Auxiliary Models

	R^B	*cv*	*.632*
F1	.02	124.64	40.84
F2	.04	56.31	19.89
F3	.12	10.03	3.51
C	.21	6.25	1.53
A1	.25	2.08	1.46
A2	.26	2.05	1.48
D	.32	0.65	2.10
B	.33	1.29	1.19
E1	.40	0.34	4.05
E2	.46	-0.69	4.66
E3	.47	-1.16	6.17
E4	.49	-3.31	5.20

Table 6: *REL, N=100*

Learning Sets The learning sets are all like model A1, except for differences in the *Euclidean distance* (δ_E) separating the two classes in $\Re^4$. The models are summarized in Table 5.

Results Table 6 summarizes the main results of Experiment III. Once again, note that these results are tabulated in order of *increasing* R^B.

The values for *REL* shown in Table 6 illustrate a clear decline in the performance of *cv* as R^B *decreases* and a corresponding decline in the performance of *.632 bootstrap* as R^B grows very large.[15]. Since *cv* appears to be the method of choice when R^B is *high* and *.632 bootstrap* the preferred method when R^B is *low*, how then does one select the appropriate technique when faced with real data for which, presumably, R^B is an unknown quantity? Since *cv* seems to do an acceptable job of selecting T_{opt} over much of range of R^B and since the *.632 bootstrap* does a superior job of

[15]This behavior is caused by a phenomenon known as *deletion sensitivity*. See Crawford (1987) for details.

error estimation *given* T_{opt}, a reasonable strategy might be to use *cv* to choose T_{sel}, *assume that* $T_{sel} = T_{opt}$, and then use the *.632 bootstrap* to estimate $R^*(T)$ for T_{sel}. This strategy was tested for the $n = 100$ trials of all twelve probability models, and the results are shown in Figure 2.

From Figure 2 it is clearly evident that the use of the combined approach (abbreviated *mix* in the Figure) minimizes the chance of poor performance when faced with either very high or very low R^B, since the combined strategy delivers results comparable to the best strategy at either extreme. In addition, it is of considerable interest and practical importance to note that for $\mathcal{L}$ exhibiting *moderate* error rate, the combined approach delivers results *better* than either strategy used alone.

2.2.5 Conclusions

Although the results of Experiments I, II and II suggest that a *combined strategy* for tree selection and error estimation will deliver impressive results, the analyst is, of course, not constrained to choosing any particular strategy. When faced with real $\mathcal{L}$, the analyst has the liberty of closely examining many candidate trees, presumably close to the tree selected by *cv*, and can therefore, with a mixture of common sense and prior knowledge of the domain, make a sensible tree selection. Once a tree has been selected, the analyst can then, if necessary, estimate $R^*(T)$.

3 INCREMENTAL CART

3.1 Introduction

Current commercial implementations of the CART methodology are not capable of incremental learning. Once a decision tree has been induced, newly acquired cases can be incorporated into the decision tree only via the brute force method of appending them to the existing $\mathcal{L}$, and then running the induction procedure from scratch on the expanded $\mathcal{L}$. The research described in this section was designed to extend the CART approach to *incremental learning*, in which newly acquired cases can be used to appropriately *update* an existing tree.

3.2 Initial Efforts

This section describes the first version of the incremental CART algorithm and, in addition, illustrates the behaviour of the algorithm with two brief examples.

3.2.1 The Algorithm

The incremental learning algorithm makes use of the fact that pairs of successor nodes in the decision tree are conditionally independent, given their immediate predecessor. The algorithm, as initially implemented, proceeds as follows: when a new case is acquired, it is sent to the existing tree and, at each node, the *new* change in impurity $\Delta\imath(s,\tau)_{new}$ that results from the *new* node population and the *existing* s_{opt} is computed (via 1). Now it is possible that, because of the acquisition of this new case, a split *different* from s_{opt} might now be optimal. It is further possible that this, as yet *hypothetical*, split might send the new case to a *different* descendant than s_{opt} currently does. In order to examine the implications of this possibility, $\Delta\imath(s,\tau)_{hyp}$, the change in impurity that would be obtained if the new case was send to the alternate descendant, is computed. If $\Delta\imath(s,\tau)_{new}$ is no worse than $\Delta\imath(s,\tau)_{hyp}$, then the new case is simply sent to the appropriate descendant. If, however, $\Delta\imath(s,\tau)_{new}$ is *worse* than $\Delta\imath(s,\tau)_{hyp}$ then a search for a *new* optimal split s_{new} of τ is initiated. If s_{new} is no different than s_{opt}, then the new case is sent to the appropriate descendant. If, however, s_{new} differs from s_{opt}, the existing descendants of τ are pruned away and the cases at τ are re-partitioned according to s_{new}, creating a pair of new descendants τ_R and τ_L. Finally, the *entire* CART procedure (including error-estimation and pruning to T_{opt}) is carried out on each new descendant.

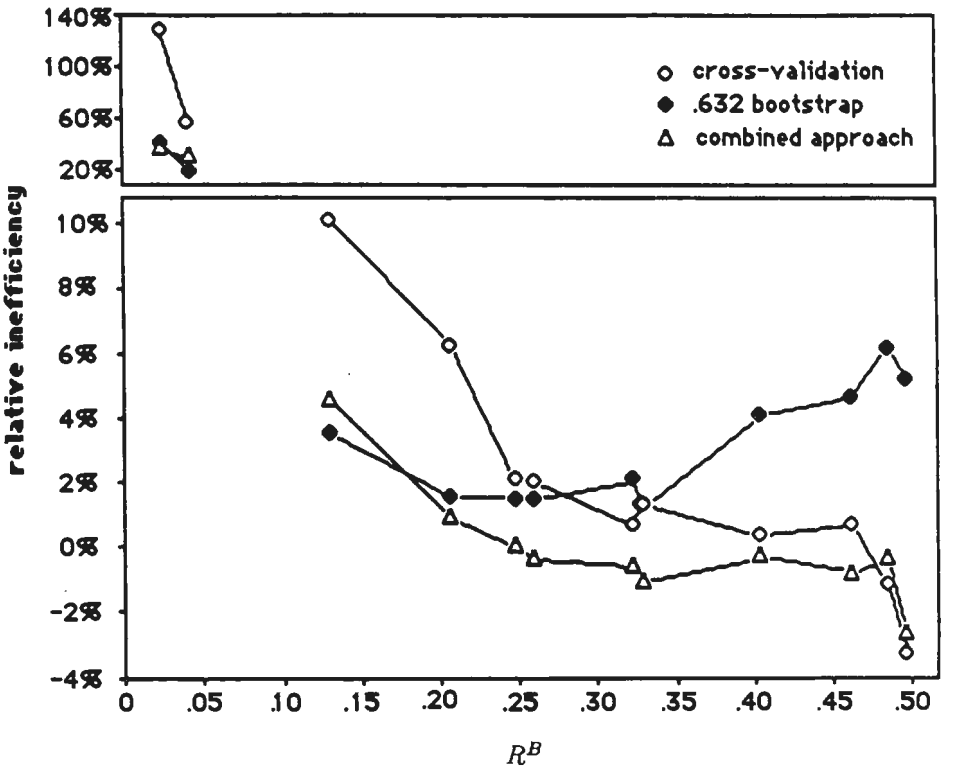


Figure 2: Results Indicating the Efficacy of the Combined Strategy

3.2.2 Initial Testing

Initial tests of the new algorithm were carried out using two different learning sets. Since it was necessary to closely monitor the behavior of the algorithm, readily interpretable learning sets had to be selected. The learning sets described in Section 1, were ruled inappropriate since one could not assign meaningful labels to their attributes. The learning sets which were eventually chosen are desribed below:

The Iris Data R.A.Fisher's classic *Iris* flower learning set (Fisher, 1936) is an ideal choice for exploring the behavior of the algorithm, largely because its characteristics are so well known. Each of the 150 cases consists of a *known* species assignment (*I. virginica*, *I. setosa*, or *I. versicolor*) and a vector of measurements of *sepal length*, *sepal width*, *petal length* and *petal width*. In order to use CART incrementally, 50 cases were chosen, at random, to be the *initial* learning set ($\mathcal{L}_{init}$), and the remaining 100 cases were assigned to the *incremental* learning set ($\mathcal{L}_{inc}$), and added to $\mathcal{L}_{init}$ one by one.

The Faulty Calculator Data This learning set[16] is attractive both for its ease of interpretability and its well defined model. Figure 3 shows the numerical display unit of a hypothetical calculator, and illustrates the components of the display which must be illuminated to generate the number 5. Now imagine that each component of the display is both *independent* of the others and *faulty*. If we let the probability that a given component *fails*[17] be ϵ, then the probability of any outcome can be easily computed as follows:

$$Prob\{(x_1, \ldots, x_7) = (1101011) \mid 5\} = (1-\epsilon)^7$$

In this way, R^B can be calculated for any ϵ. For example, when $\epsilon = 0.1$, $R^B = 0.26$. The learning set for testing the algorithm was generated

[16] First described in Breiman *et al* (1984).

[17] *i.e.* illuminates when it should not, or vice-versa.

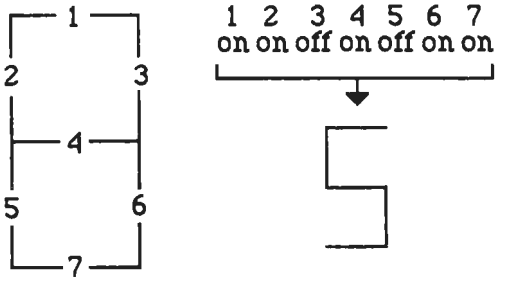


Figure 3: The Calculator Display

with $\epsilon = 0.1$ such that $\mathcal{L}_{init}$ had 50 cases and $\mathcal{L}_{inc}$ had 150 cases.

CART Calls	s_{new} *Searches*
4	7

	$\hat{R}(T)$	$\hat{R}^{cv}(T)$	N
$\mathcal{L}$	0.04	0.06	150
$\mathcal{L}_{init}$	0.04	0.08	50
$\mathcal{L}_{init} \rightarrow \mathcal{L}$	0.04	0.07	$50 \rightarrow 150$

Table 7: *Iris* data, initial algorithm.

CART Calls	s_{new} *Searches*
15	64

	$\hat{R}(T)$	R^B	N
$\mathcal{L}$	0.23	0.26	200
$\mathcal{L}_{init}$	0.18	0.30	50
$\mathcal{L}_{init} \rightarrow \mathcal{L}$	0.18	0.27	$50 \rightarrow 200$

Table 8: *Calculator* data, initial algorithm.

Results Performance of the algorithm was monitored as follows: the structure and error rate of the T_{sel} induced incrementally is compared to the error rate and structure of the T_{sel} induced from $\mathcal{L} = \mathcal{L}_{init} + \mathcal{L}_{inc}$ and, in addition, counts are kept of both the number of times that the CART procedure is invoked, and the number of times that a search for a new s_{opt} is initiated.

Although the trees induced incrementally were remarkably similar to the trees induced *en masse*, this was hardly surprising because of the very large number of tree regrowths which took place. Since the error rate of the trees induced from $\mathcal{L}_{init}$ was very close to the error rate of the trees induced from $\mathcal{L} = \mathcal{L}_{init} + \mathcal{L}_{inc}$, it was hoped that CART would be invoked infrequently and that much of the computational burden of the incremental algorithm would consist of updating node contents and evaluating $\Delta\iota(s,\tau)_{new}$ and $\Delta\iota(s,\tau)_{hyp}$. The high frequency with which CART was called was both unexpected and discouraging. Table 7 summarizes the results for the *Iris* data and Table 8 summarizes results for the *calculator* data.

Given that the error rate of the trees induced from $\mathcal{L}_{init}$ was so similar to that of the trees induced from $\mathcal{L}$, why was CART invoked so frequently? A close examination of the behavior of the incremental algorithm reveals that the frequent tree regrowing is due both to *attribute redundancy*, in which one attribute is highly correlated with one or more other attributes, and *attribute noise* in which, for a given node, the *exact* choice of s_{opt} is determined largely by random perturbations in $\mathcal{L}$. Figure 4 illustrates the pattern of tree regrowths for the *Iris* data. Note that, for each new rule induced by CART, the new case that caused CART to be invoked is described as:
`(species s-l s-w p-l p-w)`.

Close examination of Figure 4 clearly reveals the problems imposed by noise and redundancy. Note that tree 2, when viewed as partition of the axis representing petal length, is identical to tree 1 save for a split change from p-l< 25 to p-l< 26. This slight change, however, caused the entire CART procedure to begin anew, at the root, when the "offending" new case was added to the tree. In a similar manner, trees 3 and 4 are nearly identical, al-

```
Start: Tree 1; N=50

IF p-l > 48.5 THEN virginica
   ELSEIF p-l < 25 THEN setosa
     ELSE versicolor


Tree 2; N=63, (virginica 62 28 48 18)

IF p-l < 26 THEN setosa
   ELSEIF p-l < 48.5 THEN versicolor
     ELSE virginica


Tree 3; N=83, (virginica 60 30 48 18)

IF p-l < 26 THEN setosa
   ELSEIF p-l < 47.5 THEN versicolor
     ELSE virginica


Tree 4; N=143, (versicolor 68 28 48 14)

IF p-l < 26 THEN setosa
   ELSEIF p-w > 16.5 THEN virginica
     ELSEIF p-l < 49.5 THEN versicolor
       ELSE virginica


Tree 5; N=148, (versicolor 67 30 50 17)

IF p-l < 26 THEN setosa
   ELSEIF p-w > 17.5 THEN virginica
     ELSE versicolor
```

Figure 4: Tree Regrowths for *Iris* Data

though this time, CART was invoked at the interior node. Such subtle shifts in s_{opt} are surely due to noise rather than *real* changes in the nature of $\mathcal{L}$. Note also the effect of the redundancy of petal-length and petal-width in tree 4, and the effects of noise in the transition from tree 4 to tree 5. The *calculator* data[18] is not affected by the noise phenomenon described above, as any given attribute can take on only two possible values: *light on* and *light off*, but suffers greatly from redundancy. Because of the inherent redundancy amongst the digit component attributes, virtually every invocation of CART simply delivers a tree *identical* to one generated at an earlier stage in the regrowth sequence.

The problem with this initial algorithm is that there is no way to ascertain when s_{new} is *sufficiently different* from s_{opt} to warrant calling CART and when s_{opt}, although different from s_{new}, should continue to be used. This deficiency is addressed in a slightly different algorithm, described below.

3.3 A Resampling Approach

How can we determine when s_{new} is sufficiently different from s_{opt} to warrant invoking CART and carrying out the entire growth and error estimation procedures? If the probability distribution of $\mathcal{S}_\tau$, the space of all possible optimal splits at node τ, were *known* then one could resort to statistical theory and only invoke CART when s_{new} fell outside some multidimensional region in $\mathcal{S}_\tau$ defined by the mean vector and covariance matrix of the optimal splits for each attribute. Putting things within the framework of statistical inference, we have obtained *point estimates* for s_{opt}, and wish to also obtain *interval estimates*. Unfortunately, however, we have no distribution theory for CART partitions upon which we may rely in order to obtain such interval estimates. As is often the case in situations like this, we make up for our lack of theoretical knowledge with computing power, and seek nonparametric interval estimates via a resampling scheme. In this case, we choose the *ordinary bootstrap* as our resampling tool and bootstrap the splits

[18] The tree growth sequence is not shown here.

in an effort to obtain *neighborhoods of split acceptability* for the nodes. We proceed as follows: given N_τ, the population[19] of cases resident at τ, draw B bootstrap samples (with replacement) from N_τ and, for each, locate s^b_{new}. The resulting neighborhood of splits about τ represents an empirical distribution function for the optimal partitions and can be used to ascertain when s_{new} is *significantly* different from s_{opt}.

The Modified Algorithm The improved algorithm differs only slightly from the first implementation, described previously. Rather than immediately invoking CART at τ when it is discovered that s_{new} differs from s_{opt}, it is first ascertained whether or not s_{new} falls *outside* the neighborhood of "acceptable splits" as defined for τ. If no neighborhood exists, then the bootstrapping procedure described above is invoked to generate one. If s_{new} falls within the neighborhood, then s_{opt} is used as before, otherwise τ_L and τ_R are pruned away, τ is re-partitioned according to s_{new}, and CART is invoked on each new descendant, as before. The user may provide a parameter to control the size of the acceptable neighborhoods, and this parameter is simply an indicator of how many quantiles of the entire empirical distribution one wishes to exclude from the neighborhood. For example, specifying a parameter of 5%, defines a neighborhood in which the outermost 10% of the univariate distribution of s_{opt} has been excluded for each attribute. When dealing with attributes measured on a nominal scale, a slightly different approach must be taken since quantiles cannot be computed for such attributes. In such cases, the user may specify a parameter indicating which points in the distribution to be discarded. For example, the user may specify that all *singletons* in the empirical distribution be discarded, so that only those splits s^b_{new} which were selected multiple times are included in the neighborhood. The results[20] of the modified algorithm are described below.

[19] Where N_τ is defined as the population existing immediately prior to the acquisition of the newest case.

[20] All results are based upon bootstrap samples of 25, 90% neighborhoods for real-valued attributes, and singleton exclusions for nominal-valued attributes.

CART	*Searches*	*Neighborhoods*
0	15	1

	$\hat{R}(T)$	$\hat{R}^{cv}(T)$	N
$\mathcal{L}$	0.04	0.06	150
$\mathcal{L}_{init}$	0.04	0.08	50
$\mathcal{L}_{init} \rightarrow \mathcal{L}$	0.04	0.07	$50 \rightarrow 150$

Table 9: *Iris* data, modified algorithm.

CART	*Searches*	*Neighborhoods*
2	192	11

	$\hat{R}(T)$	R^B	N
$\mathcal{L}$	0.23	0.26	200
$\mathcal{L}_{init}$	0.18	0.30	50
$\mathcal{L}_{init} \rightarrow \mathcal{L}$	0.18	0.28	$50 \rightarrow 200$

Table 10: *Calculator* data, modified algorithm.

Results Tables 9 and 10 indicate the much improved performance of the modified algorithm. An acceptable error rate is maintained with dramatically reduced computational burden[21], as compared with the initial approach. It is also of interest to note that both calls to CART which were made when the *calculator* data was used were calls originating at nodes quite close to the leaves of the current tree. This is in marked contrast to the initial algorithm, in which 7 of the 15 calls to CART originated at the root.

3.3.1 Estimating Error Incrementally

In addition to maintaining tree *structure* incrementally, it is important to also maintain an error rate estimate incrementally. Although it is a straightforward task to incrementally update and maintain the $\hat{R}(T)$ estimate of error, $\hat{R}(T)$ is unacceptably biased. Maintain-

[21] Although *more* searches for s_{new} take place, such searches demand *much* less computational effort than calls to the CART algorithm, and so overall computational expense is considerably reduced.

ing $\hat{R}^{cv}(T)$ is, alas, not quite so easy, since $\hat{R}^{cv}(T)$ is obtained as a result of invoking CART, the very activity we wish to minimize. Despite this, a quite effective approach has been found which, although not unbiased, is at least a considerable improvement over $\hat{R}(T)$. The approach essentially involves traversing the tree and taking a weighted average of existing $\hat{R}^{cv}(T)$ and $\hat{R}(T)$ estimates.

The Algorithm Starting at the root, the tree is traversed and those nodes lowest[22] in the tree, for which $\hat{R}^{cv}(T)$ estimates have been computed, are marked. Now, since those $\hat{R}^{cv}(T)$ estimates were computed, a number of new cases have, presumably, been acquired at those nodes. Therefore at a given node for which an $\hat{R}^{cv}(T)$ estimate exists, we make the assumption that the new cases that have arrived subsequent to the computation of $\hat{R}^{cv}(T)$ are acting as an independent test sample[23]. Under this assumption, we compute an *updated* error estimate using an average (weighted by the number of cases) of the $\hat{R}^{cv}(T)$ and "test sample" error rates. The parent of every marked and updated node is now updated by forming a weighted average of the error rates of its children[24], and the process proceeds recursively until an updated estimate of the error rate at the root is obtained. The $\hat{R}^{cv}(T)$ values for the incrementally updated trees in Tables 7 and 9 were obtained using this algorithm. The R^B values in Tables 8 and 10, on the other hand, were obtained from a *true* independent test sample and therefore the error updating scheme was not required.

3.4 Conclusions

The incremental CART algorithm, as described here, offers considerable potential as a means by which accurate and parsimonious trees can be maintained, with minimal computational burden, in situations in which new training data is continually acquired. In order to fully validate the utility of the procedure, however, much work remains. In particular, Monte Carlo trials similar to those described in Section 2 must be undertaken in an effort to explore the performance of the algorithm for a variety of different sample sizes and probability scenarios. In addition, Monte Carlo "swindles" and efficient updating formulae need to be explored in an effort to further reduce the computational burden involved in generating neighborhoods of acceptable splits. Such work is currently underway.

[22] Where lowest means farthest from the root.

[23] This is not quite correct, since the new cases act as a *true* independent test sample at the root only. This assumption is, therefore, a source of bias.

[24] When one child is a leaf, a weighted average of the updated $\hat{R}^{cv}(T)$ and $\hat{R}(T)$ estimates is computed, otherwise a weighted average of the two updated $\hat{R}^{cv}(T)$ estimates is computed.

A Appendix

A.1 The Datasets

Datasets F1, F2, F3, A1, A2, E1, E2, E3 and E4 were generating using the RANNOR function in *SAS*©, dataset C was generated via an acceptance/rejection technique implemented in a Pascal program written by the author, dataset D was generated using the *IMSL*© subroutine GGNSM, and the *calculator* dataset was generated via a Pascal program written by the author.

A.2 Implementation

The experiments in Section 2 were all carried out using an implementation of CART written in IBM PascalVS and running on an IBM 3084 mainframe computer. The experiments in Section 3 were carried out using an implementation of CART written entirely in Franz Inc. Extended COMMON LISP.

A.3 Acknowledgements

Thanks to Bob Fung and Steve Kaye of Advanced Decision Systems for helpful and insightful comments. Special thanks to Prof. Jerome Friedman of Stanford University for teaching me everything I know about CART.

References

Breiman, L., Friedman, J.H., Olshen, R.A., & Stone, C.J. (1984), *Classification and Regression Trees*, Belmont: Wadsworth.

Crawford, S.L. (1987), *Resampling Strategies for Recursive Partitioning Classification Using the CART Algorithm.* Ph.D. Thesis, Stanford University.

Efron, B. (1982), *The Jackknife, the Bootstrap, and Other Resampling Plans*, SIAM NSF-CBMS, Monograph 28.

Efron, B. (1983), Estimating the Error Rate of a Prediction Rule: Improvements on Cross-Validation, *Journal of the American Statistical Association*, 78 (382).

Feigenbaum, E.A. (1981), Expert Systems in the 1980's, in *State of the Art Report on Machine Intelligence* (A. Bond, Ed.), Maidenhead: Pergamon-Infotech.

Fisher, R.A. (1936), The Use of Multiple Measurements in Taxonomic Problems, *Annals of Eugenics* 7 (1).

Fisher, W.D. (1958), On Grouping for Maximum Heterogeneity, *Annals of Statistics* 2 (2).

Gordon, L. & Olshen, R.A. (1978), Asymptotically Efficient Solutions to the Classification Problem, *Annals of Statistics* 6 (3).

Hunt, E.B., Marin, J., & Stone, P. (1966), *Experiments in Induction*, New York: Academic Press.

Kononenko, I., Bratko, I. & Roskar, E. (1984), Experiments in Automatic Learning of Medical Diagnostic Rules, Technical Report, Jozef Stefan Institute, Ljubljana, Yugoslavia.

Patterson, A. & Niblett, T. (1983), *ACLS User Manual*, Glasgow: Intelligent Terminals Ltd.

Quinlan, J.R. (1979), Discovering Rules by Induction from Large Collections of Examples, in *Expert Systems in the Micro Electronic Age* (D. Michie, Ed.), Edinburgh: University Press.

Therneau, T.M. (1983), A Short Introduction to Recursive Partitioning, Technical Report, ORION 021, Stanford University, Statistics Department, Stanford CA.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

MOLE: A Knowledge Acquisition Tool That Buries Certainty Factors

Larry Eshelman
Department of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

MOLE is a knowledge acquisition tool for helping experts build systems that do differential diagnosis. Diagnostic expert systems often have to rely upon inferences that involve some degree of uncertainty. Typically, the tentativeness of the rules of inference is represented by certainty factors or some other cardinal measure of support. Unfortunately, this information is difficult to acquire from the experts. This paper describes how MOLE is able to dispense with certainty factors. By integrating into its problem-solving method several heuristic assumptions about how evidence relates to hypotheses, and by including in its knowledge acquisition process a way of generalizing the expert's preferences, MOLE does not need to elicit certainty factors from the domain experts or to internally represent the degree of support of an inference rule with certainty factors. This facilitates knowledge acquisition with no loss to diagnostic performance.[1]

1. A Lack of Confidence in Certainty Factors

Experts often have to deal with uncertainty. The data they use may be noisy and the rules of inference may be uncertain. A major area of research in AI has been concerned with developing methods for handling uncertainty which are theoretically sound, yet, computationally tractable. However, the soundest, computationally tractable theory of uncertainty will be of little value if the information needed cannot be provided by the experts. This paper is concerned about handling uncertainty from the point of view of knowledge acquisition. In particular, it will discuss how a particular knowledge acquisition tool for building expert systems, MOLE [Eshelman 86, Eshelman 87], acquires knowledge from the domain expert about how to handle rules of inference that are only tentatively held without needing to elicit certainty factors.

MOLE is both a performance system ($MOLE_P$) which interprets a domain dependent knowledge

---

[1]This research was sponsored by Digital Equipment Corporation (DEC).

base and a knowledge acquisition tool ($MOLE_{KA}$) for building and refining this knowledge base. $MOLE_P$ selects or classifies candidate hypotheses on the basis of "rules" associating the candidate hypotheses with various evidential considerations. $MOLE_{KA}$ builds a knowledge base by eliciting knowledge from the domain expert, guided by its problem-solving method, and discovers missing knowledge and refines the knowledge base, guided by its understanding of how to diagnose what knowledge the problem-solving method might be missing.

MOLE belongs to a family of knowledge acquisition tools which get their power by paying close attention to the problem-solving method used by their performance systems [McDermott 86]. Examples of such systems are TEIRESIAS [Davis 82], ETS [Boose 84, Boose 87], MORE [Kahn 85a, Kahn 85b], SALT [Marcus 87], and KNACK [Klinker 87]. MOLE's predecessor is MORE. The approach taken by MOLE, especially with regard to the handling of uncertainty, can best be understood in the context of our experiences with MORE.

MORE, like MOLE, is a knowledge acquisition tool for building diagnostic expert systems whose problem-solving method is heuristic classification [Clancey 84, Clancey 85]. MORE's rules of inference are similar to those used by MYCIN [Buchanan 84]. Each piece of evidence supporting some hypothesis has associated with it a number representing its degree of support or certainty. Thus, in order to build a knowledge base, MORE needs to elicit from the domain expert a set of candidate hypotheses, evidence supporting each hypothesis, and a number representing the degree of support of that evidence.

One of the shortcomings of MORE as a knowledge acquisition tool is that experts do not like providing numeric certainty factors and are not very good at it. When experts are asked by MORE to indicate the degree of support of some piece of evidence for a hypothesis by selecting a number within a fixed range, they will think about the question for a while and then almost inevitably choose some number near the middle of the range. We first tried to remedy this problem by switching from numeric certainty factors to more "meaningful" qualitative measures of support such as "very likely", "likely", "moderately likely", "unlikely" and "very unlikely". The result was that the experts continued to spend a great deal of time worrying about which value to choose, still chose a value near the middle, but now complained about having their choice restricted to a small, discrete set of values.

One of the motivations in the development of MOLE was the elimination of the need for asking experts for support values or certainty factors. Our experience with MORE indicated that certainty factors do not need to be very accurate. We hypothesized that on the basis of a few simple heuristics MOLE could assign default certainty factors that are just as good as the certainty factors initially assigned by the expert. These heuristics made use of the structure of the evidential network. However, in order to obtain the necessary leverage, the evidential associations could no longer be arbitrary implications among arbitrary facts about the world as they are in MYCIN [Szolovits 78] and to a lesser extent in MORE. The evidential associations must be constrained by giving them a more specific, causal interpretation.

$MOLE_{KA}$ was used to build knowledge bases for a number of domains. In each case MOLE was able to assign a plausible set of default certainty factors to the network. Using these default certainty factors, $MOLE_P$ was able to perform as well as when the factors were assigned

by the experts. Of course, the $MOLE_{KA}$ assigned certainty factors often needed to be adjusted, but so did those assigned by the expert. The earlier versions of MOLE tried to adjust the certainty factors by fitting them to the desired behavior. This suggested to us that certainty factors could be dispensed with completely. Given that we are ultimately concerned with diagnostic performance, not getting the numbers rights, we hypothesized that a cardinal representation of certainty could be replaced with an ordinal representation that directly maps onto the desired behavior. This is how uncertainty is now handled by MOLE.

There is no question that a cardinal representation of an inference's degree of support potentially contains more information than an ordinal representation. MOLE compensates for this loss of representational power by assigning a more structured interpretation to the evidential associations. In the next section the evidential structures represented in MOLE will be discussed from the point of view of $MOLE_P$'s problem-solving method. In the third section, MOLE's ordinal representation of evidential preferences will be discussed. In the fourth section, the knowledge acquisition process whereby MOLE refines and generalizes these locally acquired ordinal preferences will be discussed.

2. MOLE's Method

$MOLE_P$ diagnoses a problem by first proposing a set of candidates that will cover or explain the symptoms or complaints specified by the user and then seeks information that will differentiate among these candidates. This cover-and-differentiate process is iterative. Often some of the evidence used to differentiate among the hypotheses needs to be explained or covered. Furthermore, once a hypothesis is accepted it may need to be explained by higher level hypotheses. When MOLE cannot further differentiate among the candidate hypotheses, MOLE tries to combine the viable candidates into a coherent explanation. MOLE makes certain default assumptions for doing this (e.g., given two sets of hypothesized events that explain all the symptoms, the smaller set is preferred). However, these assumptions can be overridden by domain dependent knowledge.

MOLE's problem-solving method can be summarized in terms of three fundamental knowledge roles:

- For each symptom or abnormal event propose a set of *covering* alternatives or explanations.
- Seek information that will help *differentiate* among these alternatives.
- *Combine* the viable local explanations into a coherent set.

Central to MOLE's method is the distinction between evidence that needs to be explained or covered by some hypothesis and evidence that helps differentiate among hypotheses. The former, covering knowledge, makes two basic assumptions about how supporting evidence relates to hypotheses.

1. Exhaustivity: every abnormal finding has an explanation -- i.e, some candidate hypothesis will account for it.
2. Exclusivity: explanations should not be multiplied beyond necessity -- i.e., do not accept two hypotheses if one will do.

Interpreting the exhaustivity assumption causally, it is equivalent to saying that every symptom

has a cause. MOLE assumes that it knows about the causes of the symptoms or abnormal findings that it is supposed to diagnosis. This knowledge of the potential explanations of an event constitutes covering knowledge. If a symptom is not explained by one of its covering hypotheses, MOLE assumes that it must be explained by one of its other covering hypotheses. If MOLE discovers that none of the covering hypotheses for a symptom is acceptable, then it reasons that it must be missing some covering hypothesis and tries to elicit this additional knowledge during the knowledge acquisition process.

The exclusivity assumption is basicly a version of Occam's razor: All other things being equal, parsimonious explanations should be favored. From a practical point of view the exclusivity assumption captures the rule of thumb that the explanatory events represented by the hypotheses are fairly rare, so it is unlikely that they occur simultaneously. Of course, two such events might be interrelated, but then this relationship should be explicitly represented in the knowledge base. It should be stressed that the exclusivity of the hypotheses is not logical exclusivity. The assumption of exclusivity allows MOLE to reject hypotheses for which there is no independent evidence and which are not needed to explain any symptom. In other words, once some hypothesis has been accepted as an explanation of a symptom, competing hypotheses are no longer needed and are rejected as explanations of that symptom. However, this does not exclude the possibility that the event associated with a rejected hypothesis explains some other symptom.

An important corollary follows from the exhaustivity and exclusivity assumptions: Accept the best candidate relative to its competitors -- i.e, a candidate may "win" because there is independent evidence for it or because competing candidates have been ruled out. Because a symptom must be explained by some hypothesis (exhaustivity), a candidate hypothesis can be confirmed by ruling out its competitors. And because only one hypothesis is likely to be true (exclusivity), a candidate hypothesis can be disconfirmed by providing independent evidence for one of its competitors but not it.

Two types of considerations are relevant for differentiating among a set of potential explanations for a symptom:

1. The independent acceptability or unacceptability of the covering or explanatory event or state (i.e., independent of what is being covered).
2. The dependability of the covering or explanatory connection (e.g., the strength of the causal connection).

In practice this means that a candidate explanation can be ruled out by providing reasons for rejecting the explanatory event or by providing reasons for doubting that the connection between the explaining and explained event holds. This distinction can be given a Bayesean interpretation. The former, event-qualifying knowledge, affects the prior probability of the event. The latter, connection-qualifying knowledge, affects the conditional probability of the symptom given the hypothesis. However, MOLE's interpretation is somewhat broader than this. An connection-qualifying piece of evidence, for example, can qualify any association, including event-qualifying associations and other connection-qualifying associations.

It should be noted that although three knowledge roles have been identified as central to MOLE's problem-solving method, i.e., covering, differentiating, and combing knowledge, the

same piece of information can play more than one role. This is illustrated by the schematic network in figure 2-1.

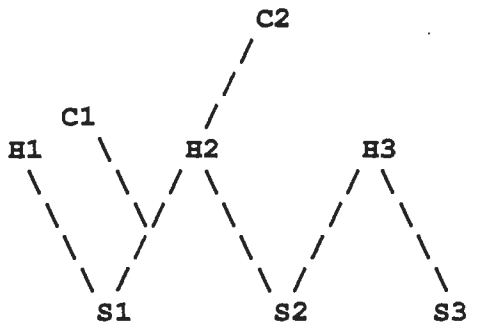


Figure 2-1:

The network consists of three symptoms, three hypotheses, and two qualifying conditions. Symptom S_1 is explained by hypotheses H_1 and H_2, and symptom S_2 is explained by hypotheses H_2 and H_3. Suppose that MOLE is faced with the task of explaining the presence of S_1 and S_2 and knows that the absence of condition C_1 is a reason for favoring H_1 over H_2 as an explanation for S_1, and that the absence of condition C_2 is a reason for favoring H_3 over H_2 as an explanation for S_2. Although locally C_1 and C_2 will have similar effects -- differentiating between the causes of S_1 and S_2, respectively -- globally their affects may be quite different. Suppose that C_1 is a connection-qualifying condition whereas C_2 is an event-qualifying condition. Both are necessary conditions, but the former qualifies the connection between H_2 and S_1, whereas the latter qualifies the event H_2. The absence of C_1 will rule out H_2 as the cause of S_2, but not rule out the occurrence of the event represented by H_2. H_2 may still occur and be the cause of S_1. Thus, knowing only that condition C_1 is absent, MOLE will accept H_1 as the explanation for S_1 and be neutral concerning the possible explanations of S_2. On the other hand, the absence of C_2 rules out the occurrence of H_2, and so will be a reason not only for favoring H_1 over H_2 as the explanation for S_1, but also for favoring H_3 over H_2 as the explanation for S_2. Thus, knowing only that condition C_2 is absent, MOLE will accept H_1 as the explanation of S_1 and H_3 as the explanation of S_2.

Conditions C_1 and C_2 illustrate how information relevant for locally differentiating among hypotheses can also play a role in helping combine the selected hypotheses into a coherent explanatory whole. Symptom S_3, on the other hand, illustrates how covering information can be used to differentiate among hypotheses. S_3, like S_1 and S_2, is an abnormal event or state whose presence needs to be explained. As the network indicates, S_3 is explained by hypotheses H_3. Suppose that the initial problem is to explain S_2, so H_2 and H_3 as explanations of S_2 need to be differentiated. Because S_3 is only explained by H_3, the presence of S_3 provides a very strong reason for believing in H_3, and thus differentiates H_3 from H_2 (in favor of H_3) as an explanation for S_2.

As a concrete illustration of these relationships, consider an example from a knowledge base built by $MOLE_{KA}$ for diagnosing automobile engine problems. One of the the abnormalities that may need to be explained is that the engine is difficult to start. There are a considerable number of potential explanations, but for the sake of illustration let us concentrate on three generic explanations: (1) a problem with the ignition system, (2) a problem with the carburator

and (3) the engine is overheating. An ignition problem can be differentiated from the other three by the absence of a spark at the spark plug. This is an event-qualifying piece of information. It is also an abnormal event which is explained by an ignition problem. The hypothesis that the engine is overheating can be ruled out by the fact that the engine has not been running recently. This is a connection-qualifying piece of information, qualifying the connection between the engine overheating and the difficulty in starting. Thus, it does not rule out the engine overheating as an explanation for other problems (e.g., damaged pistons). Furthermore, even if there is independent evidence that the engine overheats, this will not explain (at least directly) why the engine is difficult to start if the engine has not been running recently.

3. Ordinal Preferences and the Structure of Evidence

In order to dispence with cardinal measures of uncertainty MOLE needs one additional knowledge structure -- preferences [Doyle 85]. Preferences, unlike certainty factors, are ordinal. A preference for H_1 over H_2 does not indicate how strong the preference is, although it is assumed that it is great enough to justify favoring one hypothesis over another. Preferences are not restricted to binary relationships. They partition competing hypotheses or conflicting reasons into equivalence classes. Preferences usually reflect the relative likelihood of their objects, but they can take into account other factors such as cost. Preferences are context specific and can be qualified by conditions. The fact that H_1 is preferred over H_2 in one context does not mean that it is preferred in another context. Preferences do not just apply to hypotheses. They apply to any situation where there can be conflicting reasons -- e.g., reasons for and against the same hypothesis. There can also be higher order preferences -- i.e., preferences about preferences.

It would seem that preferences make the acquisition of differentiating knowledge quite easy. $MOLE_{KA}$ only needs to elicit from the expert information that will enable it to know when to favor one hypothesis over another. In other words, the expert is only required to provide, in the context of relevent evidence, an ordinal ranking of the candidate hypotheses. Although this would make the knowledge acquisition process simple, it would also make it long, tedious, and stupid. It would require that the expert rank the hypotheses for every viable combination of the evidence.

The problem with any ordinal ranking of hypotheses, as opposed to a cardinal measure of support, is that it does not provide any mechanism for combining evidence. When cardinal measures of support are used, the expert can indicate not only that he is more confident in one hypothesis than another but by how much -- e.g., twice as confident. Furthermore, not only is it possible that these measures of certainty can be combined, there are reasonably good theories available for doing this -- e.g., Bayesean theory, Dempster-Shafer theory [Shafer 76]. The fact that the application of these theories is often computationally intractable only means that in practical applications certain heuristic shortcuts may be necessary. MYCIN style certainty factors, for example, can be interpreted as one method of simplifying Bayesian theory for use in practical applications [Charniak 83].

As theoretically appealing as these numeric approaches may be there are two fundamental problems with incorporating them into expert systems. First, as I indicated earlier, it has been

our experience that experts are not very good at specifying their degree of certainty in a way that can be consistently captured by these cardinal representations. This may not simply be a matter of communication. It is been questioned whether experts make use of explicit degrees of belief in their own reasoning [Harman 86]. On the other hand, experts have little trouble ranking the candidate hypotheses potentially explaining a particular symptom under various evidential conditions. It should be kept in mind that what we are concerned about is behavior -- the diagnostic performance of the system -- not getting the numbers right. It seems that the only practical way of getting the numbers right is by fitting them to the desired behavior.

Secondly, even if the expert could provide a cardinal indication of his degree of confidence, it is not likely that any single method for combining evidence would be applicable [Cohen 83, Gruber 87]. The problem is that evidence has a structure and cannot be reduced to a single numeric measure which can be manipulated by a combination function. This is not to say that stress on structure is incompatible with a numeric approach. In fact, the "belief networks" built by MOLE structurally resemble those of Pearl's approach [Pearl 86]. This should not be too surprising. As was pointed out in the previous section, MOLE's distinction between event-qualifying evidence and connection-qualifying evidence is analogous to the Bayesian distinction between prior probabilities and conditional probabilities. However, instead of propagating numbers representing probabilities through a tree-like network, MOLE makes categorical decisions at each node in light of the evidence of the surrounding nodes. However, these decisions are always tentative and are often made on the basis of default assumptions. MOLE makes use of dependency directed backtracking in order to maintain a consistent network [Doyle 79].

MOLE is able to avoid needing a preference for every combination of evidence that might help differentiate among a set of hypotheses because it is able to generalize many of its local preferences so that they can be integrated into a richer set of evidential structures. In the next section this aspect of the knowledge acquisition process will be discussed. In the remainder of this section some of the benefits of this enriched interpretation of preferences will be illustrated.

In the previous section the importance of determining whether a piece of differentiating knowledge is event-qualifying or connection-qualifying was illustrated. Integral to deciding whether a piece of information is event-qualifying is deciding what event it qualifies. The identity of the event being directly qualified is not always obvious. A reason can indirectly support a hypothesis by directly ruling out its competitors or vice versa. Even in cases where the relevant hypotheses explain no other symptoms, the decision as to which event a piece of evidence qualifies can have a subtle effect on the results of diagnosis. This is illustrated by the schematic networks in figure 3-1. Symptom S_1 is explained by either H_1 or H_2. Suppose that it is known that reason R_1 is a reason for favoring H_1 and reason R_2 is a reason for favoring H_2. These reasons could be represented by preferences. However, by refining these preferences into event-qualifying or connection-qualifying conditions, much more is known. Consider two possible ways these reasons could be event-qualifying. In the first case, R_1 directly supports H_1 and R_2 directly supports H_2. In the second case, R_1 directly supports H_1, but R_2 indirectly supports H_2 by providing evidence against its only competitor, H_1, as an explanation for S_1. If either H_1 or H_2 explains other symptoms besides S_1, then the choice between these two representations will be quite important. Furthermore, even in this limited network, the presence of both reasons will be handled differently depending upon the representation. In the first case,

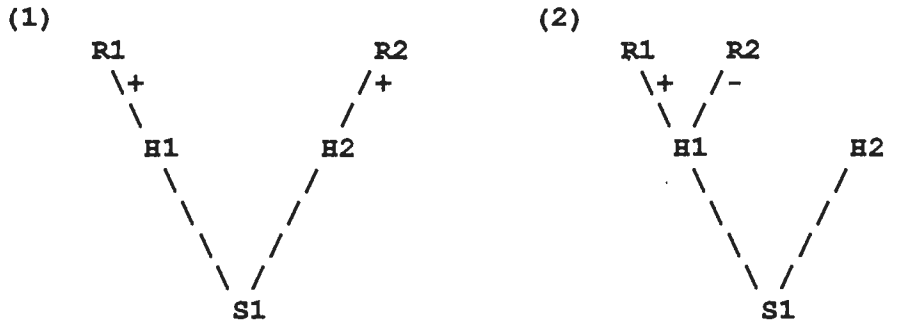


Figure 3-1:

the presence of both R_1 and R_2 is not a genuine conflict. It is simply a case where there are independent reasons for accepting both H_1 and H_2. MOLE cannot tell which is the actual cause of S_1. Both may be. But even if only one is the actual cause, both hypotheses need to be looked into. (That is, even if H_1 is the actual cause of S_1, something needs to be done about H_2 or it may soon be causing S_1.) In the second case, the presence of both R_1 and R_2 is a genuine conflict. If $MOLE_{KA}$ has not been told that it should prefer one over the other, $MOLE_P$ will simply let them cancel each other. MOLE then has no reason for believing in either H_1 or H_2, although it assumes that one must be true in order to explain S_1.

Even if some piece of differentiating evidence has been categorized as an event-qualifying condition and the event being qualified has been correctly identified, further refinement of how the reason bears on the event can be important when combining evidence. In particular, it is useful to know that a reason supports the hypothesis in question by being caused by it rather than causing it. This is illustrated by the schematic networks in figure 3-2.

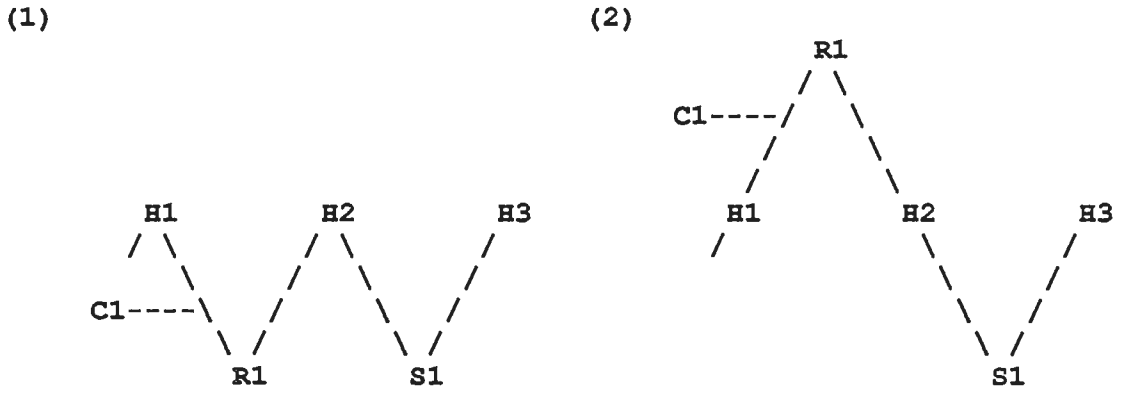


Figure 3-2:

In both networks the nodes have the same connections, but they are interpreted differently. In (1) R_1 is potentially explained by H_2 (as well as by H_1), and thus helps support H_2 over H_3 as an explanation for S_1. In (2) R_1 is a cause of H_2 (as well as of H_1), and thus supports H_2 over H_3 as an explanation for S_1. (This is represented by the way the networks are laid out -- the convention in these schematic networks is to place the cause above the effect.) In (1) if condition C_1 fails to hold and the connection between H_1 and R_1 is cut, the support of R_1 for H_2 will be enhanced. In (2), on the other hand, the failure of C_1 to hold will have no such effect, even though the supporting connection between R_1 and H_1 is still cut. In general, if a reason,

e.g., R_1, supports several hypothesized events, e.g., H_1 and H_2, because it is explained by these events, then if it is ascertained that it is explained by one of these events, its support for the other events is cancelled.

4. Knowledge Acquisition

We have seen that for MOLE a complete, usable knowledge base consists of (1) a network of symptoms and alternative explanations, (2) information for differentiating among the alternative explanations for each symptom, and (3) and knowledge as to how to combine the locally accepted hypotheses into a coherent explanation. Of the three types of knowledge, the first, covering knowledge, is the most basic and provides the foundation upon which differentiating and combining knowledge is acquired. The acquisition of covering knowledge is the most straighforward part of the knowledge acquisition process. If the problem cannot be cast systematically in terms of covering knowledge, then MOLE is probably the wrong knowledge acquisition tool.

$MOLE_{KA}$ initiates the knowledge acquisition process by asking the expert to list the possible problems or complaints that would motivate the intended user to seek help in diagnosing them. MOLE next elicits possible explanations for the complaints, and then higher level possible explanations of the explanations. The result is a multi-leveled network of symptoms and possible explanations.

The acquisition of differentiating knowledge would be quite simple if all MOLE had to do was elicit preferences. However, as we have seen, although differentiating knowledge can be represented by ordinal preferences, $MOLE_{KA}$ needs to refine at least some of these preferences into event-qualifying and connection-qualifying knowledge if MOLE is to be avoid acquiring preferences for every possible combination of evidence, and, yet, make reasonable inferences. This refinement process involves making some generalizations. $MOLE_{KA}$ proceeds by hypothesizing that a particular generalization might be in order and asks the expert a few questions to confirm its hypothesis. MOLE is guided by three principles when acquiring and refining differentiating knowledge:

1. If an expert indicates that there are multiple explanations for a symptom, then they probably can be differentiated.
2. Whenever inadequacies are discovered in the knowledge base, priority should be given to refining existing knowledge structures over acquiring new information.
3. The refinement of a piece of differentiating knowledge should not extend beyond its operational significance.

In the following three subsections these principles will be elaborated upon.

4.1. Static vs. Dynamic Analysis

The first principle is balanced by the third principle. $MOLE_{KA}$ seeks enough information to make sure that all hypotheses explaining any symptom can be differentiated, but it does not seek any more information than will minimally meet this requirement. By examining the network MOLE can determine whether it is possible for the hypotheses explaining a symptom to be differentiated in principle. If some of the hypotheses are not differentiable, MOLE will elicit enough information to meet this requirement. However, there is no guarantee that this

information is adequate to in fact differentiate the hypotheses in an actual diagnostic session. Some of the evidence required may not be available, or it may have the wrong "value" -- e.g., something is absent, when it needs to be present in order to help differentiate. $MOLE_{KA}$ has no way of knowing what information will be available until it is faced with an actual test case. There is little point in trying to cover all the possibilities during the initial acquisition of the knowledge base. Often, the expert cannot provide this needed information, unless he is reminded of it by an actual example. Therefore, MOLE's refinement of its knowledge base consists of two phases. During the first phase, it statically examines the knowledge base to make sure that all hypotheses can at least be differentiated in principle. During the second phase, the expert provides $MOLE_P$ with a test case and tells $MOLE_{KA}$ the correct diagnosis. MOLE dynamically compares its diagnosis with that of the expert. If its diagnosis is incorrect, MOLE tries to determine the source of the errors and recommends possible remedies. MOLE uses the context provided by the test case to help constrain its search for possible remedies.

4.2. Exploiting Existing Knowledge

We have already seen how neighboring covering knowledge can be used to help differentiate a shared hypotheses from its competing hypotheses. This is a direct use, in a differential role, of information that was acquired as coverering knowledge. Differentiating knowledge can also be indirectly linked with covering knowledge. Covering knowledge sanctions inferences from symptoms to hypotheses. MOLE recognizes a special kind of differentiating knowledge, anticipatory knowledge, which sanctions inferences from hypotheses to symptoms or, conversely, sanctions inferences from the absence of symptoms to the falsehood of hypotheses. Although anticipatory knowledge is closely associated with covering knowledge, it is independent of covering knowledge. Anticipatory knowledge is dependent upon the strength of the causal connection between a hypothesis and a symptom. Covering knowledge says that a hypothesized event might be a cause of a symptom. Anticipatory knowledge says that a hypothesized event, if it occurs, almost always causes a particular symptom. The fact that a hypothesized event causes a symptom does not mean that the symptom is likely to be present if the hypothesized event occurs. In other words, one cannot infer from knowledge of the occurrence of the hypothesis to the presence of the symptom. To make this clear, consider the extreme case where the hypothesized event is the only explanation of some symptom. It follows that if it is known that the symptom is present, then it can be inferred that the hypothesis is true. However, this does not license the inference from knowledge about the occurrence of the hypothesis to the anticipated presence of the symptom. The causal connection between the hypothesis and the symptom may be tenuous. Perhaps certain enabling conditions must be true before the symptom can be caused by the hypothesized event. The additional knowledge that a covering event typically causes some symptom to be present is a special form of differentiating knowledge which we will refer to as anticipatory knowledge. Given the hypothesized event, the symptom is anticipated. Conversely, if the symptom is not present, there is a good reason for ruling out the hypothesized event.

The intimate relationship between covering and anticipatory knowledge is exploited in knowledge acquisition. Existing covering knowledge provides a good place to first look for differentiating knowledge. As an illustration of MOLE's exploitation of covering knowledge to discover differentiating knowledge consider another example from the automobile diagnosis

domain. Two abnormalities that need to be diagnosed are (1) excessive mechanical noise from the engine and (2) lack of power. MOLE has been told that the first symptom, excessive mechanical noise, is explained by (a) worn crankshaft bearings, (b) worn pistons, or (c) worn cylinders and that the second symptom, lack of power, is explained by the same set of hypotheses as well as an additional hypothesis, (d) valve problems. MOLE needs to be able to differentiate among these hypotheses for both symptoms. $MOLE_{KA}$ proceeds by asking the expert whether the absence of excessive mechanical noise from the engine will tend to rule out any of the events (a), (b) and (c). The expert informs MOLE that this will rule out (a), worn crankshaft bearings. MOLE next asks the expert whether the absence of lack of power will tend to rule out any of the events (a), (b), (c), and (d). The expert informs MOLE that this will rule out worn pistons, (b), and worn cylinders (c). Now $MOLE_P$ can at least potentially differentiate worn crankshaft bearings from worn pistons and cylinders. If there is excessive mechanical noise from the engine, but no lack of power, MOLE will conclude that the crankshaft bearings are worn. On the other hand, if there is lack of power but no excessive mechanical noise, then MOLE will be able to rule out one of the candidate hypotheses explaining lack of power, worn crankshaft bearings.

4.3. Testing Generalizations

If $MOLE_{KA}$ is not able to exploit existing information to discover differentiating knowledge, it will request from the expert new information that will help it differentiate among the hypotheses in question. When MOLE acquires a new piece of differentiating knowledge, it initially represents it as a preference. However, MOLE always asks the expert whether this new piece of information is an abnormal event or state state that needs to be explained. If it is, then MOLE will ask for potential explanations. In this case, MOLE is able to associate the new differentiating knowledge with covering knowledge, and so can try to use this new structure to guide it in refining the differentiating knowledge beyond a rudimentary preference.

When there is no usable covering knowledge to exploit, MOLE's task is somewhat more difficult. It must decide whether to treat the new information as a simple preference or try to refine it. The third principle says that there is no point in trying to refine knowledge beyond its operational significance. For example, if a symptom S_1 is explained by two hypotheses, H_1 and H_2, and no other symptoms are explained by these hypotheses, and the only differentiating knowledge is a preference for H_1 in the presence of E_1 and a preference for H_2 in its absence, then it will not be worth trying to figure out whether E_1 is an event or connection qualifying reason. This information will have no impact on diagnosis. As a rule of thumb, to the extent that covering associations differentiated by a set of preferences are interconnected, the more useful it is to try to refine the preferences. $MOLE_{KA}$ proceeds by forming a hypothesis about how a preference can be refined (and thus generalized). It then tests this hypothesis by asking the expert a few questions. If the hypothesis is confirmed MOLE replaces the preference with a richer knowledge structure, otherwise it will form and test another hypothesis. If there are no more testable hypotheses, it will abandon the attempt at refinement of the preference.

Preferences partition competing hypotheses or reasons into equivalence classes. One of the clues that MOLE uses in hypothesis formation is the relative sizes of these equivalence classes. If, for example, some piece of information partitions the set of hypotheses explaining some

symptom into two equivalence classes -- those that are to be rejected and those that are viable -- and if one set is considerably larger, MOLE will hypothesize that the reason directly qualifies the hypotheses in the smaller set. That is, if R_1 is a reason for favoring H_1 and H_2 over H_3, then MOLE first hypothesizes that the reason directly provides evidence against H_1.

MOLE also is guided by the structure of the network when generating hypotheses. The schematic network in figure 4-1 can be used to illustrate how MOLE generates and tests hypotheses.

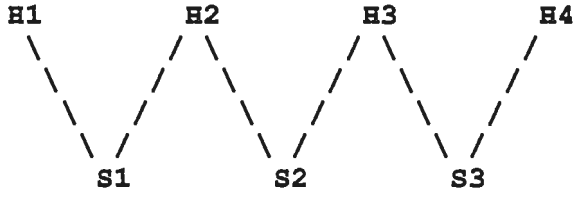


Figure 4-1:

Suppose that S_2 is present, and the expert indicates that R_1 (not shown) is a reason for preferring H_2 over H_3 as an explanation for S_2. MOLE will consider three hypotheses:

1. R_1 directly supports H_2.
2. R_1 directly rules out H_3.
3. R_1 is a condition that cuts the link from H_3 to S_2.

MOLE tests the event-qualifying hypotheses first, since they are usually easier to confirm. In this case, the candidate explanations are partioned into two equivalence classes of equal sizes, so MOLE does not have any reason to suppose R_1 directly supports H_2 rather than directly rules out H_3 or vice versa. MOLE will arbitrary pick one of the hypotheses and test it. MOLE will test the hypotheses by asking the expert hypothetical questions. For example, to test the first hypothesis, MOLE will present the expert with a situation where he needs to explain the presence of S_1, and ask him whether R_1 would be a reason for favoring H_2 over H_1. If this hypothesis is confirmed, then MOLE knows that R_1 is a reason for favoring H_2 as the explanation not only of S_1 and S_2 but other symptoms that it might be known to potentially explain.

The conditions for confirming each of the three hypotheses are summarized below:

1. If R_1 is also a reason for preferring H_2 over H_1 when S1 is true, then R_1 is assumed to be directly supporting H_2.
2. If R_1 is also a reason for preferring H_4 over H_3 when S3 is true, then R_1 is assumed to be directly ruling out H_3.
3. If there is an independent reason for believing in H_3, but because of R_1, H_2 is preferred to H_3 as an explanation for S_2, then R_1 is assumed to be a condition that cuts or disables the connection from H_3 to S_2.

If none of the hypotheses are confirmed, MOLE will continue to treat R_1 as a local preference favoring H_2 over H_3 as an explanation for S_2.

As a concrete illustration of this process, let us return to the problem of differentiating among the causes of lack of power in an automobile engine. $MOLE_{KA}$ has learned that the absence of excessive mechanical noise from the engine tends to rule out the worn crankshaft bearings hypothesis. However, in a test case where there was lack of power, but no usual mechanical

noise from the engine, $MOLE_P$ was unable to differentiate the worn crankshaft bearings hypothesis from the other hypotheses explaining lack of power. After being informed that it should have ruled out this hypothesis, $MOLE_{KA}$ asked the expert to provide information that would help differentiate among the hypotheses. The expert informed MOLE that a relevant piece of information is that the air conditioner was turned off. If the air conditioner is turned off, then the worn crankshaft bearings hypothesis is significantly less favored than the other hypotheses. At this point MOLE only knows that it should prefer the other three hypotheses over the crankshaft hypothesis in light of this new evidence. MOLE does not know how this evidence affects its evaluation. However, MOLE realizes that it is important that it better understand how this information effects its evaluation since the hypotheses in question do not just explain these two symptoms but others as well -- e.g., the engine misfiring. MOLE guesses that because only the crankshaft hypothesis was adversely affected relative to the other hypotheses, the new information is event-qualifying, and in particular, it directly provides negative evidence against the crankshaft hypothesis. MOLE tests this hypothesis by asking the expert whether knowledge that the air conditioner is turned off will affect the evaluation of the explanations for excessive noise coming from the engine. MOLE expects that it will tend to rule out the crankshaft hypothesis. However, the expert responds that it will have no effect. In the light of this reply, MOLE revises its hypothesis. It now hypothesizes that the fact that the air conditioner is turned off is a condition-qualifying piece of evidence, and in particular, that it is a condition for worn crankshaft bearings explaining the lack of power. MOLE asks the expert to assume that he has independent evidence that the crankshaft bearings are worn, and asks whether he would be inclined to believe that this is also the cause of there being a lack of power given that he knows that the air conditioner is turned off. The expert says "no" and MOLE's second hypothesis is confirmed.

$MOLE_{KA}$ makes use of this "what-if" technique during both static and dynamic analysis. During static analysis its goal is to make sure that the hypotheses are differentiable in principle. All neighboring evidential connections are considered relevant for the purposes of differentiation. During dynamic analysis $MOLE_{KA}$'s goal is to refine its knowledge base so that $MOLE_P$ will produce the diagnosis specified by the domain expert. The task is constrained by the information collected during diagnosis. If $MOLE_{KA}$ identifies the problem as being due to an incorrection differentiation or combination of hypotheses, as opposed to lacking a relevant covering connection, MOLE will proceed by asking "what-if" questions as it does during static analysis.

5. Conclusion

$MOLE_P$ is able to deal with uncertainty without needing a cardinal measure of support by integrating ordinal preferences into a structured notion of evidence. $MOLE_{KA}$ avoids a combinatorial explosion of preferences by making use of the evidential structure to help it generate and test hypotheses about how to generalize and integrate the local preferences into a more structured evidential network. There is no guarantee that $MOLE_{KA}$ will be able to generalize enough locally acquired preferences so that $MOLE_P$ will know how to combine evidence in most of the situations that it has never been explicitly told about. However, as a rule of thumb, whenever the evidential structure is sparse and offers little guidance for refining the preferences, the preferences are also sparse and so generalization is not that important.

$MOLE_{KA}$ has been used to re-build the knowledge bases originally built with the version of MOLE that used certainty factors, as well as to build knowledge bases for several new domains. In none of these domains has MOLE needed to acquire an excessive number of preferences in order to combine evidence. On average, preferences make up less than ten percent of the knowledge base. The elimination of certainty factors has enabled MOLE to take a more flexible, responsive approach to the knowledge acquisition process without sacrificing the performance of the resulting expert system.

Acknowledgements

Many people have made important contributions to MOLE. Foremost is John McDermott who helped shape the direction of the project from its inception. I have also benefited from useful discussions with Jon Doyle, Georg Klinker, Herve Lambert, Sandra Marcus, Dan Offutt, Isabelle Paoli, and Jos Shreinemakers. I would also like to thank Jos Schreinemakers for serving as MOLE's expert in the automobile diagnosis domain.

References

[Boose 84] Boose, J.
Personal construct theory and the transfer of human expertise.
In *Proceedings of the National Conference on Artificial Intelligence.* Austin, Texas, 1984.

[Boose 87] Boose, J., and J. Bradshaw.
Expertise Transfer and Complex Problems: Using AQUINAS as a Knowledge Acquisition Workbench for Expert Systems.
International Journal of Man-Machine Studies 26(1):3 - 28, to appear 1987.

[Buchanan 84] Buchanan, B. and E. Shortliffe.
Rule-based Systems: the Mycin experiments of the Stanford Heuristic Programming Project.
Addison-Wesley, 1984.

[Charniak 83] Charniak, E.
The Bayesian Basis of Common Sense Medical Diagnosis.
In *Proceedings of the National Conference on Artificial Intelligence.* Washington, D.C., 1983.

[Clancey 84] Clancey, W.
Classification problem solving.
In *Proceedings of the National Conference on Artificial Intelligence.* Austin, Texas, 1984.

[Clancey 85] Clancey, W.
Heuristic classification.
Artificial Intelligence 27, 1985.

[Cohen 83] Cohen, P. and M. Grinberg.
A theory of heuristic reasoning about uncertainty.
AI Magazine 4(2), 1983.

[Davis 82] Davis, R. and D. Lenat.
Knowledge-Based Systems in Artificial Intelligence.
McGraw-Hill, 1982.

[Doyle 79] Doyle, J.
A truth maintenance system.
Artificial Intelligence 12, 1979.

[Doyle 85] Doyle, J.
Reasoned assumptions and Pareto optimality.
In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 87 - 90. Los Angles, 1985.

[Eshelman 86] Eshelman, L. and J. McDermott.
MOLE: a knowledge acquisition tool that uses its head.
In *Proceedings of the National Conference on Artificial Intelligence.*
Philadelphia, Pennsylvania, 1986.

[Eshelman 87] Eshelman, L., D. Ehret, J. McDermott, and M. Tan.
MOLE: a tenacious knowledge acquisition tool.
International Journal of Man-Machine Studies 26(1):41 - 54, 1987.

[Gruber 87] Gruber, T. and P. Cohen.
Design for Acquisition: Designing Knowledge Systems to Facilitate Knowledge Acquisition.
International Journal of Man-Machine Studies 26(2):143 - 159, 1987.

[Harman 86] Harman, G.
Change in View: Principles of Reasoning.
The MIT Press, 1986.

[Kahn 85a] Kahn, G., S. Nowlan, and J. McDermott.
Strategies for knowledge acquisition.
IEEE transactions on Pattern Analysis and Machine Intelligence 7(5), 1985.

[Kahn 85b] Kahn, G., S. Nowlan, and J. McDermott.
MORE: an intelligent knowledge acquisition tool.
In *Proceedings of Ninth International Conference on Artificial Intelligence.*
Los Angelos, California, 1985.

[Klinker 87] Klinker, G., C. Boyd, S. Genetet, and J. McDermott.
A KNACK for Knowledge Acquisition.
In *Proceedings of Sixth National Conference on Artificial Intelligence.*
Seattle, Washington, 1987.

[Marcus 87] Marcus, M.
Taking Backtracking with a Grain of SALT.
International Journal of Man-Machine Studies , to appear 1987.

[McDermott 86] McDermott, J.
Making expert systems explicit.
In *Proceedings of 10th Congress of the International Federation of Information Processing Societies.* Dublin, Ireland, 1986.

[Pearl 86] Pearl, J.
Fusion, Propagaton and Structuring in Belief Networks.
Artificial Intelligence 29(3), 1986.

[Shafer 76] Shafer, G.
A Mathematical Theory of Evidence.
Princeton University Press, 1976.

[Szolovits 78] Szolovits, P. and R. Patil.
Categorical and probablilistic reasoning in medical diagnosis.
Artificial Intelligence 11, 1978.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

KNOWLEDGE ACQUISITION IN A SECURE ENVIRONMENT

N. E. Fredman
Planning Research Corporation
1500 Planning Research Drive
McLean, VA 22102

Abstract: As DoD and civilian agencies begin to use expert systems to perform classified tasks such as sensor fusion or strategic analysis, knowledge engineers are required to practice their art in a secure environment. Regardless of the knowledge acquisition methodology, allowances must be made for the sensitivity of the data that may be collected. Because knowledge acquisition is based upon the free exchange of knowledge, this may prove fatal to the endeavor. Security restrictions thus provide a new set of hurdles to the knowledge engineer. However, knowledge acquisition, such as that based on structured interviews, can be executed in such a way that these restrictions are circumvented legally. This paper addresses practical matters of knowledge acquisition in the face of such necessary obstacles.

INTRODUCTION

As agencies of the Government discover that expert system technology may be made to work for them, the demand for implementing this technology using classified data increases along with demands for the more usual unclassified applications. Classified expert systems applications include such things as sensor fusion for strategic and tactical intelligence usage, prediction of the movements of strategic targets (Buteau, et al., 1985), intelligence message interpretation (Loatman and McCown, 1987), and even the development of software security monitors. All of these applications require the knowledge engineer to work with data sources and human experts whose knowledge is classified.

Performing knowledge acquisition that can not only create a knowledge structure out of human know-how, but a structure that is consistent, testable, and maintainable, requires considerable effort. What then are the consequences of adding restraints upon where knowledge acquisition can be performed, who (or what) may do it, and even what the expert is allowed to tell the knowledge acquirer? In short, working in a secure environment makes a hard task even harder. For this reason, many professionals may choose to avoid the problem altogether and eschew classified jobs. Others of us find that not only is work in classified environments a normal part of our business, but that many of the problems that we are asked to confront are excellent opportunities for working out truly knotty problems in cognitive science and expert systems technology.

Whatever methodology is used to build an expert or knowledge based system in a classified environment must be flexible enough to work around security restrictions. Therefore, to provide context for discussion of practical problems, this paper first outlines a knowledge acquisition methodology based upon the structured interview. The discussion then draws a picture of the practical problems attending the secure environment, followed by a sampling of solutions to them.

A MANUAL KNOWLEDGE ACQUISITION METHODOLOGY

There are a number of alternative methods for practicing knowledge acquisition (Hoffman, 1987; Prerau, 1987) that are all valid in the appropriate situations. Since the focus of this paper is upon practical matters that may profoundly affect technical performance, rather than the particulars of the technical performance itself, discussion of which methodology is more appropriate will for the most part be left to other forums. The methodology described here (Fredman, 1986) has been used, with variations, in the Moving Target Analysis prototype (described in Buteau, 1985) and the even more highly classified Relocateable Targeting Experiment, both performed at Planning Research Corporation. Therefore this methodology serves as an adequate example for exploring what happens to knowledge acquisition when security restrictions are involved.

Stages of Knowledge Acquisition

This methodology is performed in three overlapping stages, which will be shown from a systems engineering point of view, as methodologies are usually required by the consumers of classified expert systems to fit into a general systems life cycle. The three stages are preparation and interview, process evaluation and knowledge base development, and system concept and architecture development.

Preparation and interview. Based upon a first-cut "paper" knowledge base derived from informal interviews and available documentation, the knowledge engineer targets areas for exploration in a structured interview. Interviews dealing with unclassified subjects are recorded to provide a verbal protocol (Ericcson and Simon, 1984), and then transcribed into a prescribed format for analysis. The transcriptions are analyzed first to structure the content, then to evaluate it to develop heuristics. The analysis can be manual or computer-assisted; either way, it is done in such a way as to provide traceability to the original utterances. The process of interview and analysis is repeated to fill in information gaps that the knowledge engineer finds. Heuristics at this stage are still English-language, and the knowledge base that is incrementally constructed gradually proceeds from concept to data structures. Figure 1 shows a schematic of how this process may proceed.

Process evaluation and knowledge base development. A healthy skepticism is a valuable trait in an engineer: it encourages one to be sure that the process that has been documented is actually the one that is required, and that it does work. Therefore we recommend evaluating the process that has just been recorded as heuristics (Naughton, 1985). Ideally, this would be done before coding into software. The intent is to judge the process without language- and machine-dependent complications imposed during implementation. Evaluation can be as simple as taking a set of heuristics back to the expert and trying them out on a difficult case to see if they work and are consistent with what the expert knows. Having seen it written out, the expert may want to adapt or refine the process as described. Thus, the interview or case-solving process may begin another iteration.

The knowledge base can be constructed in parallel with the evaluation, giving an opportunity to begin the prototype early. However, until the problem-solving methods which will appear in the knowledge base are identified (perhaps such as Chandrasekaran (1986) describes) the form of the knowledge base and the control structures should remain loose. The old engineering adage of form following function is advised, because much of what the knowledge engineer

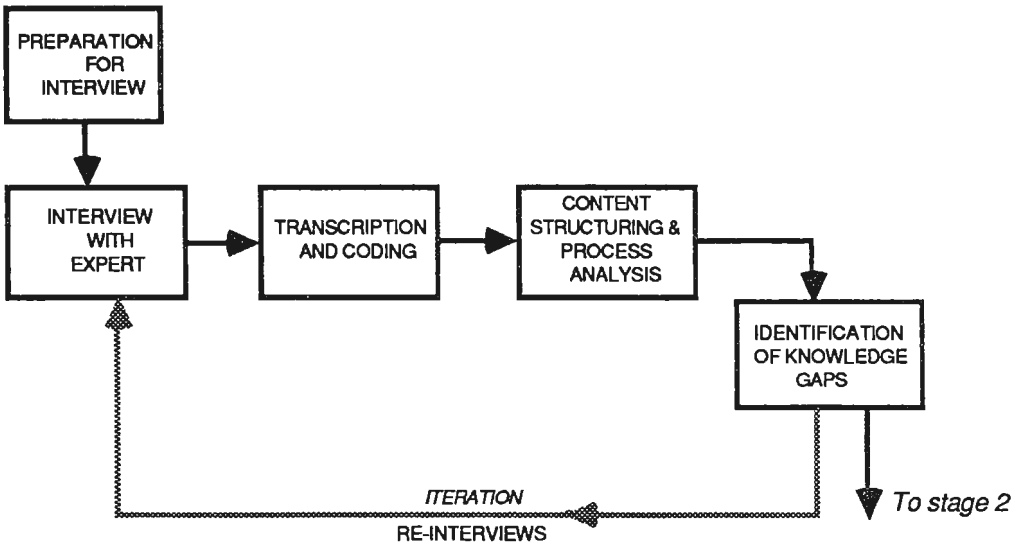


Figure 1. Stage 1-- Preparation and Interview

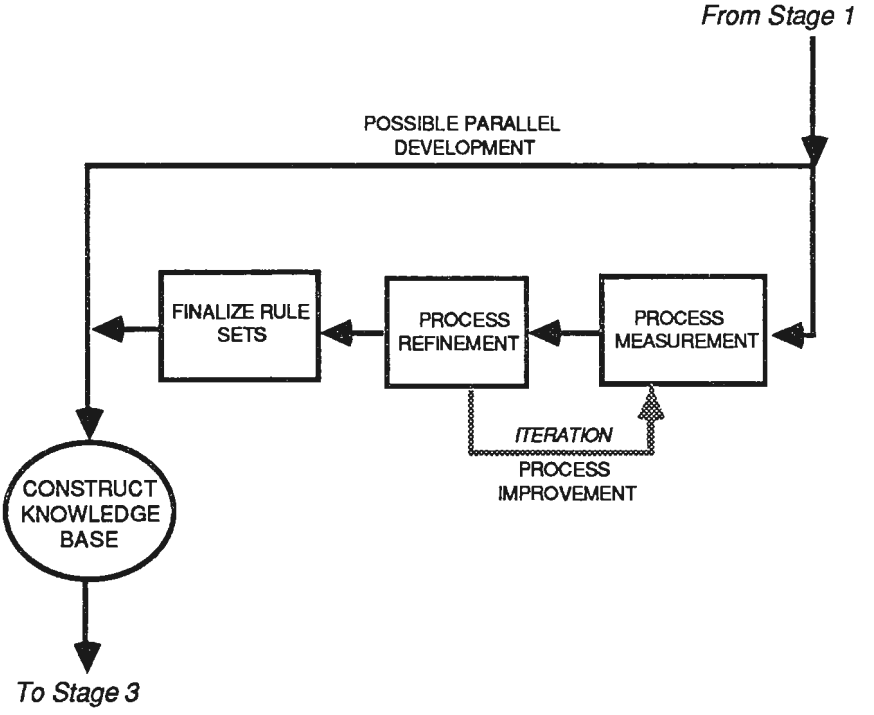


Figure 2. Stage 2 -- Process Evaluation and Knowledge Base Development

collects from the expert is useful in determining the functional system design. Figure 2 show the general direction of process evaluation.

System concept and architecture development. To develop the system concept and architecture, the knowledge engineer executes a series of cross-mappings between the functional process (represented in the knowledge base and control structures), the customer's requirements, present capabilities, and desired evolutionary system characteristics, as shown in Figure 3. The customer's needs, the available software and hardware capabilities, and the problem-solving knowledge itself are fitted together at this high level to create as firm a foundation as possible for the implementation of the prototype. The architecture is developed in such a way that the system is truly a developmental prototype, put into use as soon as there is software and the requirements are satisfied.

Traceability and Adaptability

This knowledge acquisition methodology emphasizes traceability, meaning that elements of the knowledge base can be traced explicitly to the expert. This requires careful documentation throughout knowledge acquisition:

- Interviews are transcribed and recorded.
- Process analysis utilizes transcriptions.
- Analysis of the transcribed material is methodical.
- Evaluations and fine-tuning use the same kinds of methodological stepping as the first-stage analysis.
- Knowledge base and architecture are created in explicit reference to requirements and expertise.

All these steps toward traceability are taken because accuracy is necessary in creating a justifiable knowledge base. Recording the interview is critical to this process, especially since memory is selective—"chunking" takes place in note-taking. Audio-visual recording captures not only word but gesture, which may show the nuances of reasoning; plain audio recording is acceptable, but lacks the capability for recording gesture. Thus, audio-visual recording is best if it can be arranged. Recording, with or without video, can be impartial, unselective, and fully accurate, which is why transcription from recordings is emphasized to such an extent in manual knowledge acquisition.

Manual knowledge acquisition has a benefit to the knowledge engineer in that it can be applied regardless of the software and hardware that is or is not available for the prototype—a practical consideration in low-budget proof-of-concept projects. Manual knowledge acquisition may also be adapted more easily to the classified environment.

ACQUIRING CLASSIFIED INFORMATION

We differentiate between two categories of classification, one that is general (GENSER), including classifications such as Confidential, Secret, and Top Secret, and Sensitive Compartmented Information, or (SCI). Much Government AI work involves SCI data. What happens when the knowledge engineer tries to collect information that is classified either GENSER or SCI?

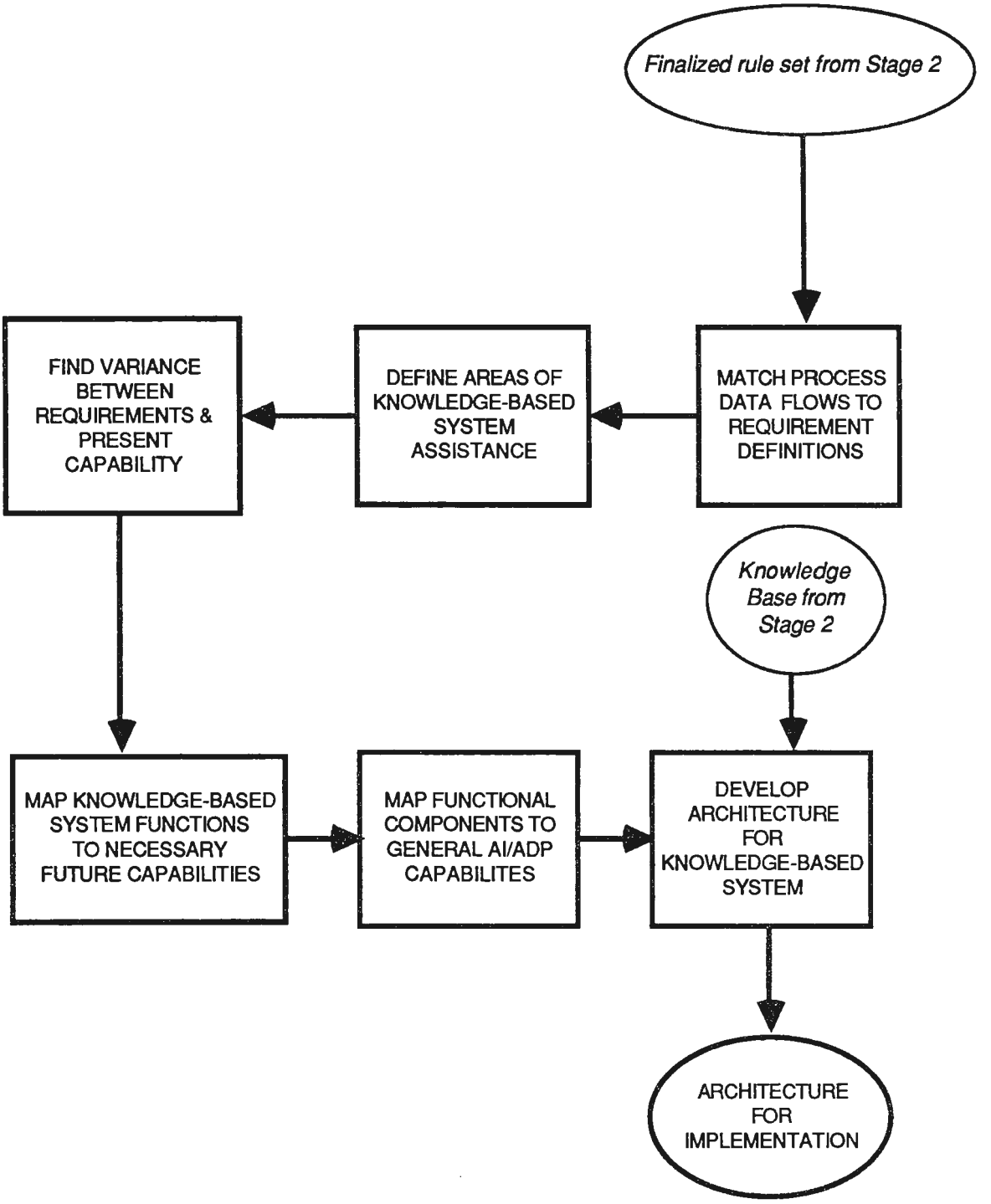


Figure 3. Stage 3 -- System Concept and Architecture Development

SCI Security Restrictions

Physical security restrictions vary with the sensitivity of the data that is being handled. There are five categories of restrictions that a knowledge engineer needs to address when preparing to work with information of this high sensitivity:

- *Personnel clearances* must be obtained for everyone who will have access to the data that will be collected, transcribed, and put into the knowledge base. Obtaining clearances for those who do not have them typically costs thousands of dollar, and may take up to two years, since background investigations are conducted.

- Clearances are not freely allotted. There are a limited number of allocatable SCI clearances. Such allocations are attached to organizations that will need them, and are available to individuals only while they are being used, within time limits. This matter of allocation, which applies to all clearances, is called *billeting*. Billets are typically expensive to maintain, as well as being scarce, and must be obtained before access to SCI material is given.

- *Need-to-know* must be established for personnel on the expert system project: clearances may be given, but access to data is not, unless it is justified.

- *Data* is carefully controlled throughout its entire life, from generation to disposal. This applies to tapes and diskettes as well as to working papers, note scraps, and finished paper documents.

- Restrictions on *recording equipment* and *computers* are quite severe, as will be discussed below. The first four restrictions are immutable, but this one, while it can create difficulties, can at least be managed.

Recording the interviews is the weak link in the knowledge acquisition methodology, and the worst case is in collecting SCI data:

> Government or contractor owned mission essential electronic equipment may be permitted within the [SCI] facility; however, this electronic equipment must be subjected to technical tests for clandestine surveillance devices and technical security hazards. If this equipment is used to process SCI [data] it will meet national standards for TEMPEST. When a technical security or TEMPEST hazard exists, [the Government] must be informed of the hazard and corrective action taken to neutralize or eliminate it. (DIA, 1980)

Because of all these constraints, security officers tend to refuse permission to record within secure spaces.

All of these restrictions have particular ramifications. First, the knowledge engineer must specify to the responsible security officers exactly how the recorder will be used. This implies that the logistical details of the structured interview must be known well in advance. Second, the security officers must certify that the recording device is actually a tape (A/V) recorder—and not a transmitting device. Third, the knowledge engineer must designate the use of the tapes. Fourth, their place of storage as well as the analysis method used must be planned ahead of time. Tapes, transcriptions, and resulting documents must be controlled as classified documents. This means that every document, including scrap paper, must be accounted for in the usual fashion, requiring extra time and attention. Finally, all the work of interviewing,

recording, transcribing, and analysis must be conducted within vaulted areas, and all computer or recording equipment therein is permanently dedicated to SCI use. It may come out of the vault—only to go into another vault. This dedication of equipment may have significant impact upon projects with limited resources. Thus, built-in delays in project working time can accrue for many non-technical reasons, such as for certifying recording equipment, computer equipment, and personnel, as well as finding space in typically cramped vaults to both store and work on the expert systems project.

GENSER Security Restrictions

Although restrictions concerning personnel clearances, billeting, and need-to know still apply, working with GENSER data classified up to Secret level is not as restricting as SCI for a variety of reasons. First, recording is allowed for material up to Secret level, providing need-to-know is examined, and sessions take place in a sound-proof room. Tapes and resulting documents are still controlled as classified documents, but need only a certified safe for storage instead of a safe in a vault when they are not being worked on. Nonetheless, what was to be an open exchange of information in this new context becomes restricted, and starts costing far more than an unclassified project in terms of built-in delays and other stumbling blocks.

In addition to these physical needs outside of ordinary technical concerns, if the expert had to work up a process before, now he has to describe a process that meets security restrictions. This can mean that while considering how to do something, or while demonstrating a technique, the expert must censor what he says to fit the restrictions. As knowledge engineers, we can only hope that he does not leave out some necessary piece of knowledge.

Circumventing the Problem—Legally

Fortunately, there are legal answers to these obstructions so that we neither compromise data nor abandon a perfectly good interview-analysis technique. In the case of SCI, the material is usually ephemeral in nature—specific information which is only true for a short period of time is what requires the SCI classification. It is, for example, not the fact that a satellite is in orbit, but what that orbit is today, that is highly classified, and that orbit may change. Likewise, not the photo-interpretation process, but today's specific photographs and their particular interpretations are SCI.

Additionally, the point of classification is often very specific, such as the name of a target being tracked, or the names of the places that it passes through. It is obvious that the names themselves contribute nothing to the process, since a it is the logical way in which ideas or objects are manipulated that is of value. Therefore, the process can be separated from the *classified* data, in many instances, and demonstrated to the knowledge engineer at a lower level of classification. The knowledge engineer and the expert may discuss nominative data, as this kind of renamed data is called, and keep their discussions of data to unclassified, confidential, or secret GENSER levels where physical security restrictions are not so onerous. Safeguarding precautions must still be taken at the Secret level, but these, as has been discussed, are not as demanding as at the SCI level: an improvement. For example, in working on PRC's MTA project, we found that we could separate the method of projecting location along a constrained physical network using scarce, uncertain data, from the real data the process was to deal with. This separation could be accomplished without loosing any of the effectiveness of the expert process.

However, there are two subtle problems beyond that of simply separating out classified data from the possibly unclassified process:

1. *Aggregation*—parts of the system may not be highly classified, but the *whole* may be. In the case of PRC's MTA, neither the nominative data, nor the embedded knowledge based system, nor the non-predictive utilities involved were by themselves highly classified, but the combined capabilities when demonstrated as a mature prototype could be classified.

2. *Integration across compartments*—expert systems to integrate solutions (such as in distributed problem solving) can have higher classifications than their components, in a case of super-aggregation. More importantly, the knowledge engineer must have the clearances to create the integrated product, so that s/he can operate in more than one SCI compartment, because crossing compartments raises the level of the necessary access clearances.

These two synergistic actions constitute a rule-of-thumb for information systems which makes them unlike hardware or machinery: classification tends to rise as work progresses. When the Triton nuclear submarine was being built, it became harder and harder to hide the existence of something the size of the Washington Monument, so classification of the overall project decreased, while the particulars remained protected. Information systems tend to work in the reverse. Data may be classified, an inference engine unclassified, and a set of rules relatively innocent, but their combined function can become an important and closely held strategic capability.

Sometimes the knowledge engineer can't keep the material at GENSER levels by a strategy of separating out data from process. Even in these cases, there are some practical alternatives in conducting interviews and analyses. First, the knowledge engineer may want to preserve accuracy, and so take the trouble to arrange audio-visual or recordings, just accepting the delays, and negotiating for cooperation from the client's security officers. Whether or not permission to record is granted, the knowledge engineer may want to use two interviewers to cross-examine the expert(s) to introduce a redundancy in memory and note-taking that may make up for any lack of recordings. A third alternative is to use a cleared stenographer to record, since it is easier often to admit people than machines to SCI facilities.

There may also be automation alternatives that involve computers, some of which can be shielded against emanations (TEMPESTed), and others which can be secluded. When voice-recognition programs can handle two people, we may use an automated stenographer: An IBM PC with a Kurzweil voice recognition program may become an alternative. Structured interviews may also be used in conjunction with automated knowledge acquisition programs (personal construct theory is one popular example) that can be dedicated to use in a secure environment. Another automated assist would be a Computer-Assisted Knowledge Engineering aid, which is essentially a transcript processor that can be implemented on the prototype "destination" machine in the secure space. Such an aid would help with with analysis and clerical necessities in the manual KA methodology, potentially reducing the number of people needed to analyze or look after a paper knowledge base because it makes the analysis process "paperless."

Regardless of mitigating measures used, the knowledge engineer should plan for delays because of control and transportation procedures that accompany knowledge acquisition under security restrictions.

CONCLUSION

Knowledge acquisition in a secure environment provides some obstacles, mostly of a non-technical nature, that the knowledge engineer must overcome. They consist of physical restrictions that often prevent use of recording equipment for developing a verbal protocol from a structured interview and restrictions on where knowledge acquisition may be conducted, and who may participate. Experts may not be able to discuss highly classified data, so may censor their discussions, which can hobble an interview.

After defining the knowledge acquisition methodology to be used, the knowledge engineer can maneuver around security constraints through divide-and-conquer separation of secure topics, alternative methods of interviewing, or automated assistance to knowledge engineer for pre-architecture formalizations.

REFERENCES

Buteau, B.L., Fredman, H., and Fredman, N.E. (1985) *Moving Target Analysis.* Technical Report TR-RD-85-1 (SECRET). McLean, VA: Planning Research Corporation.

Chandrasekaran, B. (1986) "Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design," *IEEE EXPERT*, 1(3)

Defense Intelligence Agency. (2 May 1980) Manual 50-3. "Security: Physical Security Standards for Sensity Compartemented Facilities." Washington, DC: Headquarters, Defense Intelligence Agency.

Ericcson, K.A., and Simon, H.A. (1984) *Protocol Analysis: Verbal Reports as Data.* Cambridge, MA: MIT Press.

Fredman, N.E. (1986) "Knowledge Acquisition in Prototyping Knowledge-Based Systems." Technical Report TR-RD-86-1. McLean, VA: Planning Research Corporation.

Hoffman, R.R. (1987.) "The Problem of Extracting the Knowledge of Experts." *AI Magazine* 8(2)

Loatman, R.B. and McCown, M.G. (1987) "Information Extraction from Natural Language Messages." To be published in proceedings of *Expert Systems in Government Conference.* Washington, D.C.: IEEE.

Naughton, M.J. (1985) "Knowledge Acquisition for Expert Systems." Seminar presentation. McLean, VA: Expert-Knowledge Systems, Inc.

Prerau, D.S. (1987) "Knowledge Acquisition in Expert System Development." *AI Magazine* 8(2)

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

ADVANCED EXPERT SYSTEM SUPPORT ENVIRONMENTS

Brian R Gaines
Knowledge Science Institute, University of Calgary
Calgary, Alberta, Canada T2N 1N4

Abstract: Tools and techniques for knowledge acquisition for knowledge based systems need to be integrated with the overall system and not treated as separate components. The variety of sources of knowledge, representations and applications within the system, and user roles, makes such integration complex. The complexity raises many issues not generally thought of as part of knowledge acquisition yet fundamentally significant in extending and integrating acquisition tools. This paper gives a preliminary analysis of the structure of knowledge-based systems as it relates to the integration of acquisition. It is intended that this structure be used as a basis for system design and it will be extended to form a 'requirements specification' for knowledge based systems integrating acquisition with other aspects of their operation.

INTRODUCTION

At the time of the first AAAI Workshop on Knowledge Acquisition for Knowledge Based Systems in Banff, November 1986, the literature on knowledge acquisition was sparse, scattered in conference publications, and difficult to access. The 1986 Workshop and the ensuing journal publication of some forty papers representing the state of the art world-wide changed that situation. The inclusion of sessions on knowledge acquisition in many conferences, the first European Workshop on Knowledge Acquisition for Knowledge Based Systems in London and Reading, September 1987, and the second Banff Workshop in October 1987, have changed the situation even further. We now have a world wide network of communicating researchers aware of one another's activities and an open, widely disseminated and readily available literature on knowledge acquisition.

The problem now is not so much to access research and experience in knowledge acquisition but to make sense of the diverse and wide ranging material available and, in particular, to apply the results to improve effectiveness in the development of knowledge-based systems. There are a number of major impediments to such understanding and application:

- A diversity of techniques and tools that overlap in their applications and where it is not clear whether they are competitive alternatives or complementary partners
- Lack of variety, detail and evaluation in the case histories of applications of the techniques and tools
- Lack of access to the techniques and tools outside the narrow research communities originating them
- Lack of standardization in the knowledge representation resulting from the techniques and tools making it difficult to integrate them and to interface them to existing knowledge based systems
- Lack of standardization in the forms of data required by the techniques and tools making it difficult to apply them in the same situation and compare them
- Lack of standardization in the user interfaces to the interactive tools making it difficult to integrate them in an effective environment for human-computer interaction
- Lack of portability in the run-time environments required by the tools making it difficult to integrate them with other systems

These are not unreasonable problems at this stage of development of knowledge based systems. We are in the midst not only of the evolution of a rapidly developing technology but also in a revolution of our understanding of the nature of knowledge and knowledge processes in society and technology. Premature standardization resulting in over-rigid systems would be inappropriate and a major impediment to progress. However, we have to balance the need for integration, dissemination and application of knowledge acquisition techniques and tools against the dangers of rigidity. This paper addresses the issues involved, the evolutionary trends, the key requirements for integration and the practical approaches that may be taken.

TRENDS IN KNOWLEDGE ENGINEERING

The basic model for knowledge engineering has been that the knowledge engineer mediates between the expert and knowledge base, eliciting knowledge from the expert, encoding it for the knowledge base, and refining it in collaboration with the expert to achieve acceptable performance. Figure 1 shows this basic model with manual acquisition of knowledge from an expert followed by interactive application of the knowledge with multiple clients through an expert system shell.

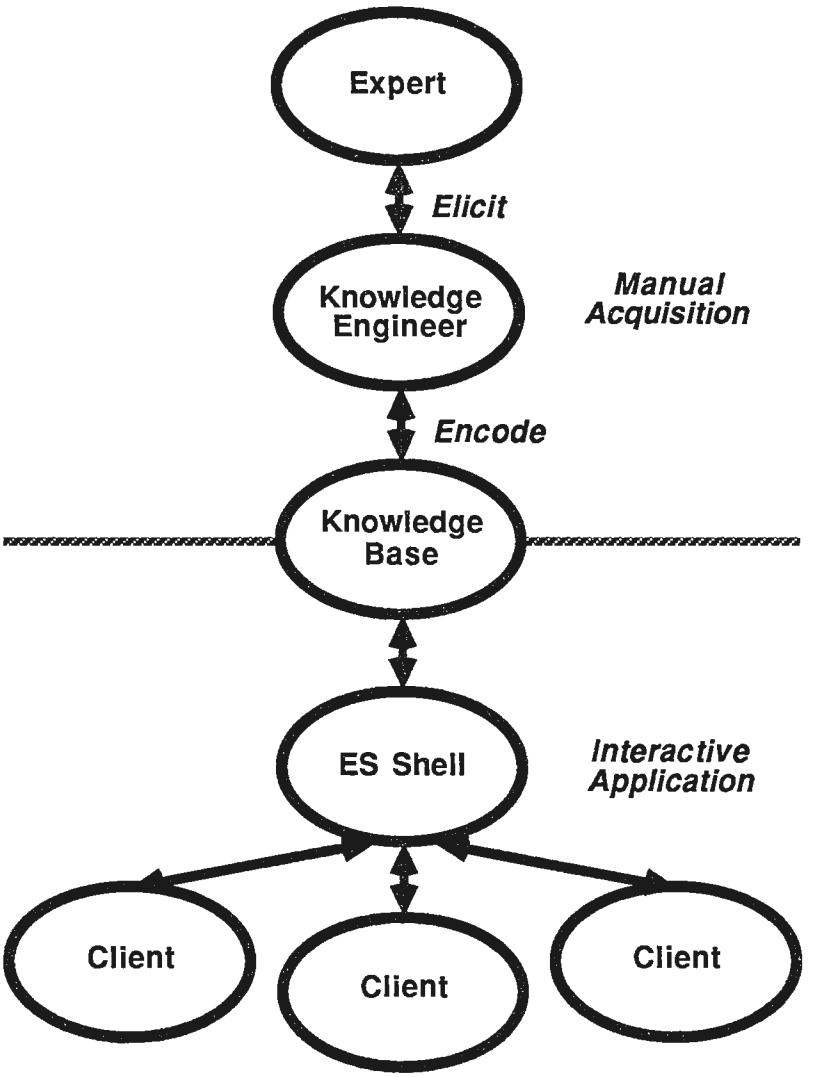


Fig.1 Basic model of manual knowledge acquisition

This basic model has been greatly extended by the introduction of interactive knowledge acquisition and encoding tools allowing the expert to transfer knowledge directly to the system without an intermediary. Such tools can greatly reduce the need for the knowledge engineer to act as an intermediary, but, in most applications, they leave a substantial role for the knowledge engineer. As shown in Figure 2, knowledge engineers can:

- Advise the expert on the process of interactive knowledge elicitation
- Manage the interactive knowledge acquisition tools, setting them up appropriately
- Edit the unencoded knowledge base in collaboration with the expert
- Manage the knowledge encoding tools, setting them up appropriately
- Edit the encoded knowledge base in collaboration with the expert
- Validate the application of the knowledge base in collaboration with the expert
- Train the clients in the effective use of the knowledge base in collaboration with the expert by developing operational and training procedures

This use of interactive elicitation can be combined with manual elicitation and with the use of the interactive tools by the knowledge engineer rather than, or in addition to, the expert. The knowledge engineer can:

- Directly elicit knowledge from the expert
- Use the interaction elicitation tools to enter knowledge into the knowledge base

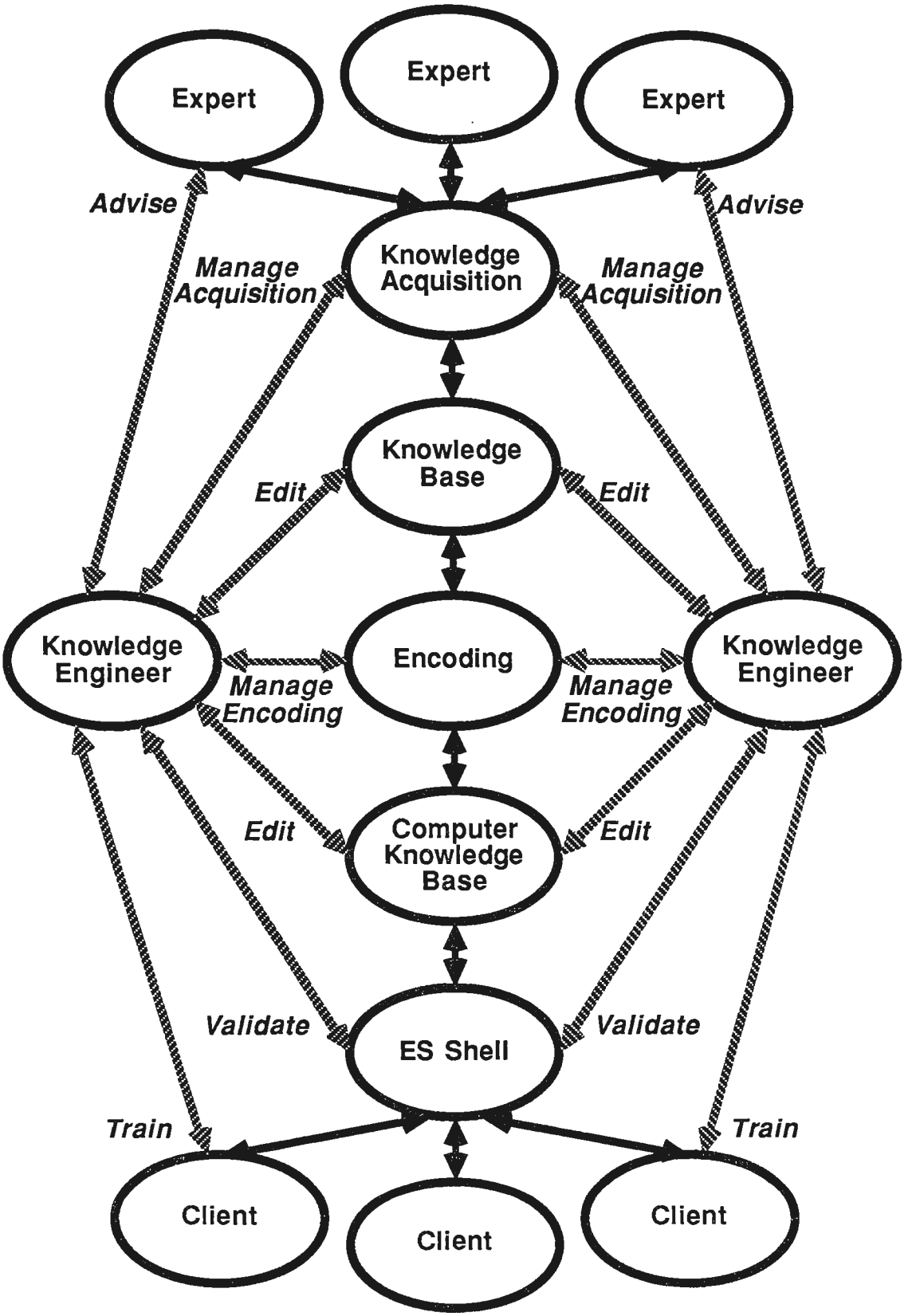


Fig.2 Knowledge engineers' roles in interactive knowledge acquisition

Figure 2 shows multiple knowledge engineers since the tasks above may require the effort of more than one person, and some specialization may be appropriate. Multiple experts are also shown since it is rare for one person to have all the knowledge required, and, even if this were so, comparative elicitation from multiple experts is itself a valuable knowledge elicitation technique.

Figure 2 shows the complexity of the knowledge engineer's role and the some of the support tools required. Figure 3 groups the support tools for editing, display, encoding and validation of the knowledge bases into an *Expert System Support Environment* (ESSE), and lumps the various forms of knowledge base together. It shows the overall structure of a knowledge-based system as a central knowledge base interacting through the knowledge acquisition tools, the expert system support environment, and expert system shell, with a user community of experts, knowledge engineers and clients.

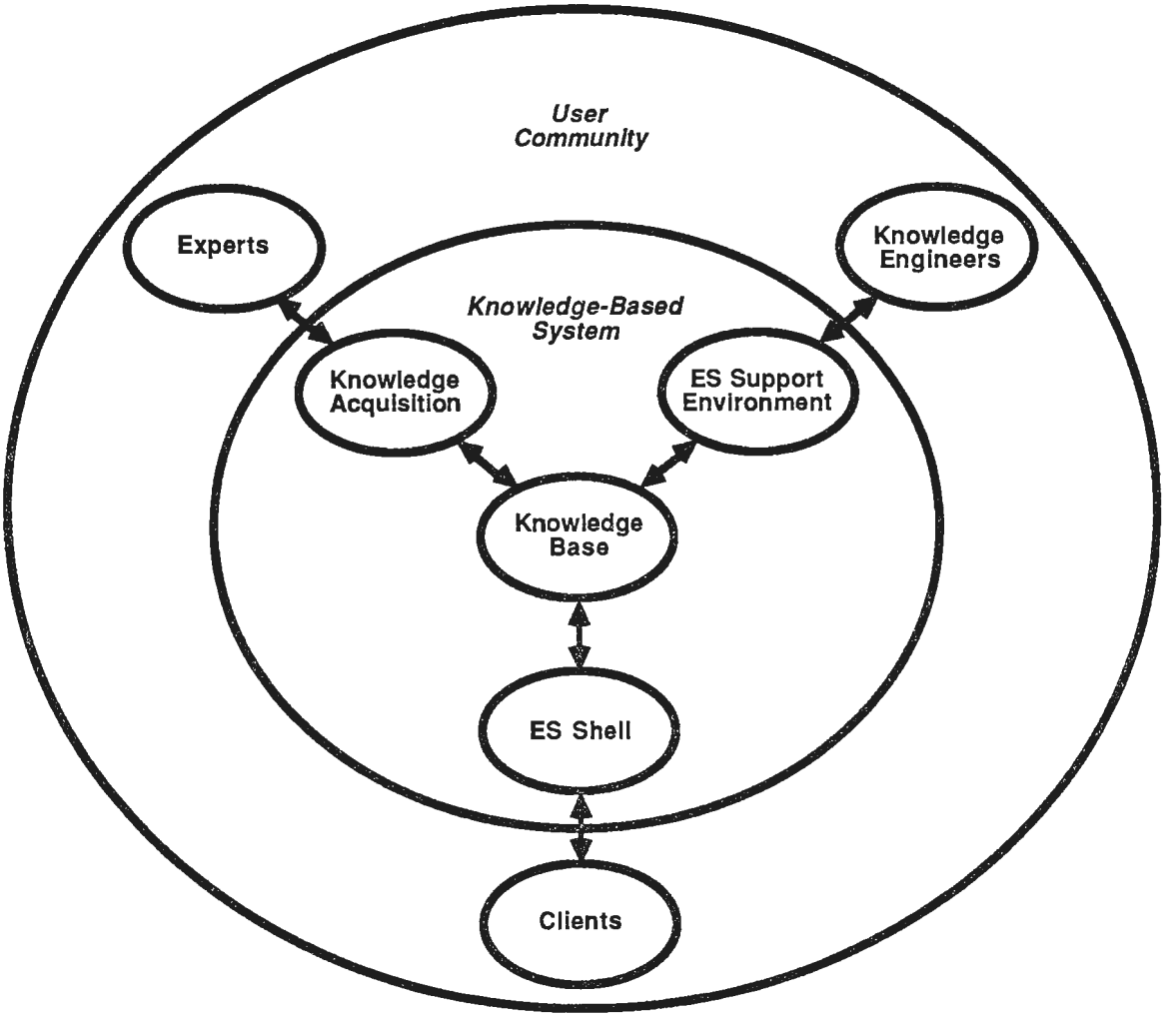


Fig.3 Major components of a knowledge based system

The notion of a user community is important in drawing attention to another trend in knowledge-based systems, that the distinction between 'experts', 'knowledge engineers' and 'clients' is often one of variable roles rather than hard and fast job definitions. The expert may be a major client for the system, or may play a major part in knowledge engineering, or both. Some clients may customize or update the system in the role of knowledge engineers or extend its capabilities in the role of experts.

INTEGRATION ISSUES IN THE USER COMMUNITY

The analysis of Figures 1 through 3 and the discussion of the previous sections raises a number of major integration issues in knowledge acquisition for knowledge based systems. Figure 4 shows these issues in terms of the major distinctions involved.

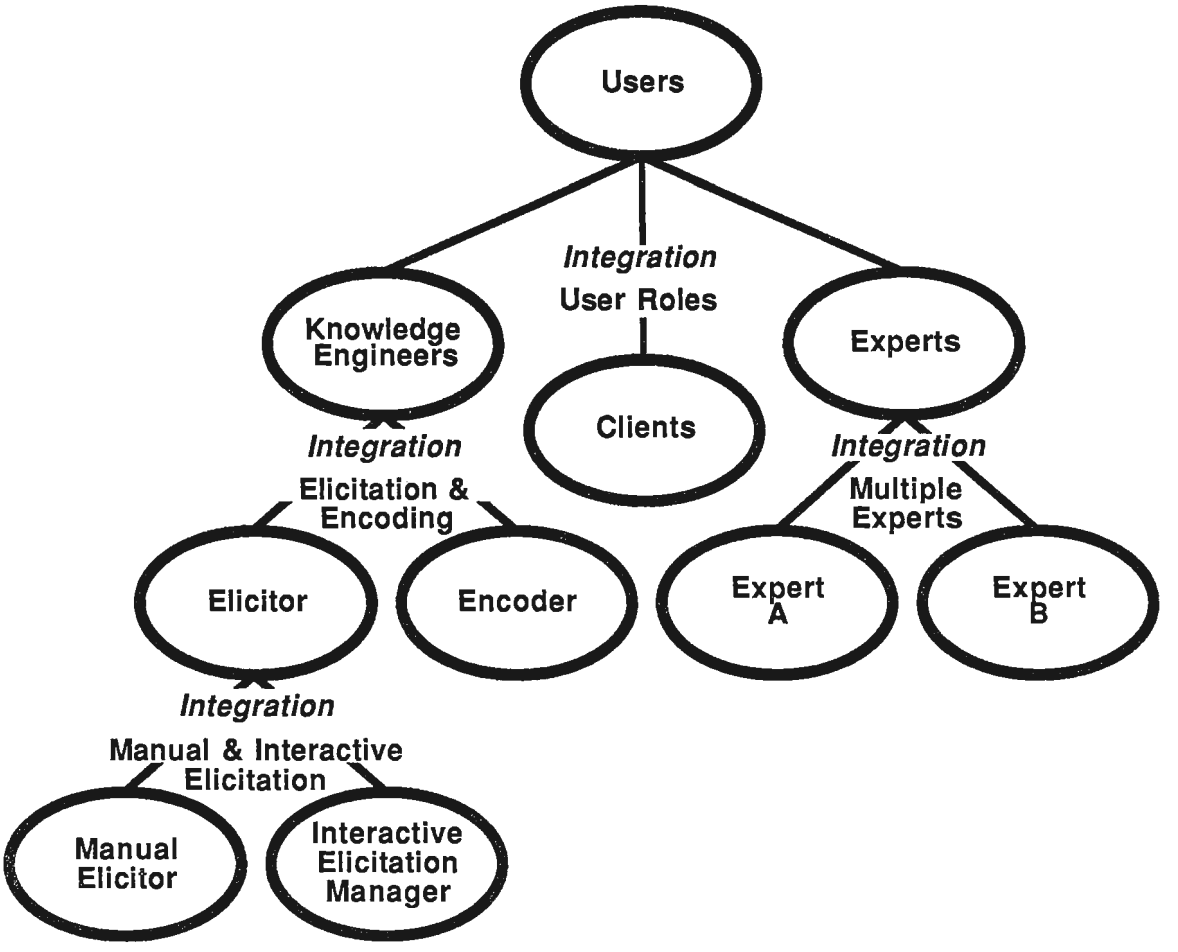


Fig.4 Major distinctions in user roles

In most current systems the knowledge acquisition tools, support tools and applications interface were designed separately and operate in very different ways on the assumption that they will be used by experts, knowledge engineers and clients, respectively. However, this diversity can be very confusing if users are flexible in their roles and switch between them. In addition, the complexity of the knowledge engineering role suggests that a variety of specializations will develop in building complex systems that will result in further roles and role switching. This leads to the design requirement:

Integration of User Roles: The user interface to a knowledge based system should be uniform and consistent across all user roles including expert, knowledge engineer and client.

However, it is important to note some features of the separation of roles:

Separation of Expert and Client Roles: The user interface to the clients should take into account the need to ensure that they use the terms in the same way as the expert. This may involve training procedures or the use of a different vocabulary.

The switch from manual to interactive elicitation has focused attention on the significance of the encoding process previously carried out tacitly by the knowledge engineer. Indeed, one of the problems in manual knowledge engineering may have been confusion between the knowledge representation meaningful to the expert and that usable by the expert system shell. A strong case has been made in manual elicitation for the use of mediating knowledge representations that are meaningful to the expert with encoding for the shell being treated as a separate phase (Johnson 1987). Such a separation is analogous to that between high level languages and machine code in conventional data processing. It may be seen as a weakness of current shell support environments that a sufficiently high level of knowledge representation is not available. The transition from Lisp code through OPS5 production rules through frames to generalized objects may be seen as the evolution of higher level knowledge representations.

The sheer variety of possible representations of knowledge that are meaningful to people makes it unlikely that the gap between a natural representation in elicitation and a computable one after encoding will ever be completely closed. Hence, as before, design requirements capturing both integration and separation are required:

Integration of Elicitation and Encoding: The knowledge representation processible by the shell in a knowledge based system should be as directly meaningful as possible to an expert.

Separation of Elicitation and Encoding: Intermediate knowledge representations should be used in elicitation for a knowledge based system if the shell representation is not directly meaningful to an expert.

The knowledge engineer's interviewing role is significant even when interactive elicitation is being used. The knowledge engineer gains information in discussion that may be used to advise the expert in the use of the tools, used to set up particular tools for elicitation, or loaded directly into the system. The system should support these various processes where appropriate:

Integration of Manual and Interactive Elicitation: Make provision for the knowledge engineer to use knowledge elicited from the expert in a variety of ways to enhance the interactive elicitation process.

And it should not be assumed that completely automatic elicitation is always possible:

Separation of Manual and Interactive Elicitation: The reduction in load on knowledge engineers through interactive elicitation should be used to enable them to be more effective in ensuring that knowledge not captured interactively is still transferred to the system.

Elicitation from multiple experts raises many problems of dealing with different vocabularies, contradictions, and so on. The system must make provision, as does AQUNIAS (Boose & Bradshaw 1987) both for integrating diverse sources of knowledge and for making contradictory sources of advice available if required. The design requirements are:

Integration of Multiple Experts: Provide tools for analyzing and expressing similarities and differences between different experts' knowledge to enable it to be combined when similar and a consensus to be negotiated when different.

Separation of Multiple Experts: When a consensus is not reached the system should be able to use contradictory sources of knowledge in generating advice by showing dissenting opinions.

The distinctions shown in Figure 4 can be extended further. For example, there may be major differences between clients making a separation between interfaces appropriate. However, the major issues relevant to knowledge acquisition are captured in the design considerations discussed.

INTEGRATION ISSUES IN THE EXPERT SYSTEM SUPPORT ENVIRONMENT

The significance of the expert system support environment shown in Figure 3 in providing tools for the knowledge engineer is apparent in the discussion above. The quality of this environment is very important in the development of expert systems, particularly in supporting the interaction between knowledge engineers and experts. Figure 5 shows some of the major issues in the ESSE in terms of the distinctions involved.

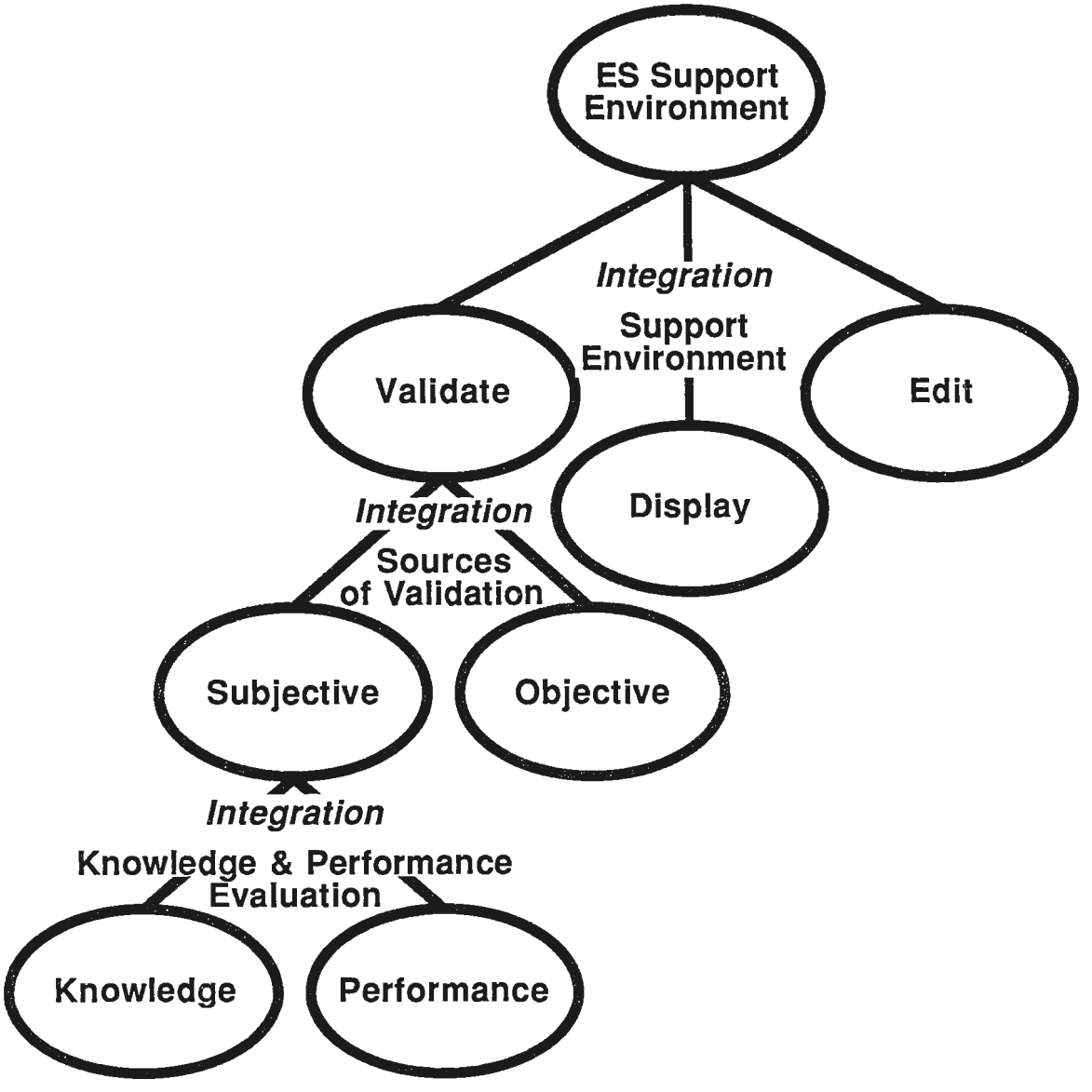


Fig.5 Major distinctions in the expert system suppport environment

The basic tools required are those for editing, displaying and validating the knowledge base. It is important that these tools present a uniform and consistent view of the

knowledge base and that they aid consistency maintenance in the knowledge base during editing. The design requirements are:

Integration of Support Environment: Provide tools for editing, displaying and validating the knowledge base which are mutually self-consistent and, if possible, automatically propagate edits to maintain consistency or, otherwise, show clearly where parts of the knowledge base have become inconsistent.

The limited converse is that different perspectives should continue to be available.

Separation of Support Environment: Support whatever variety of perspectives on the knowledge base is necessary to ensure users in different roles have access that they find meaningful.

Validation is a complex topic in its own right and it is important that a variety of tools are provided to support different forms of validation. In particular, the objective validation of the knowledge base in terms of its performance and its subjective validation in comparison with experts should be supported. The design requirements are:

Integration of Sources of Validation: Provide tools for validating the knowledge base both in terms of its absolute performance and in comparison to experts.

Objective validation may require significant case histories that can be collected only from an operational system in actual applications.

Separation of Sources of Validation: The tools for subjective and objective validation may need to be substantially different and operate over different timescales.

There is a natural separation in subjective validation between experts' direct agreement with the knowledge base and correspondence between their behavior and that of the system based on the knowledge base. Both types of validation should be supported. The design requirements are:

Integration of Knowledge and Performance Evaluation: Provide tools for enabling experts to evaluate the knowledge base directly and for comparing system and expert behavior in advising clients.

It is possible for there to be contradictions between these two forms of evaluation.

Separation of Knowledge and Performance Evaluation: Take into account that there may be contradictions between experts' evaluations of knowledge and the resulting correspondence in performance.

INTEGRATION ISSUES IN KNOWLEDGE ACQUISITION

The significant features of the knowledge acquisition component of Figure 3 are best analyzed by considering the natures of the classical and expert system design paradigms. The center column of Figure 6 shows a system for which a decision or control system is to be designed together with sources of knowledge about the system.

The left hand column shows the use of these sources in the classical system design paradigm:

- Knowledge of past case histories is used to select a class of system models
- The information required to discriminate within this class determines how the system should be instrumented for data acquisition
- Data is collected from the system through the instrumentation
- A model is identified from the model class which best fits the data
- This model is used to design a decision or control system for optimal performance

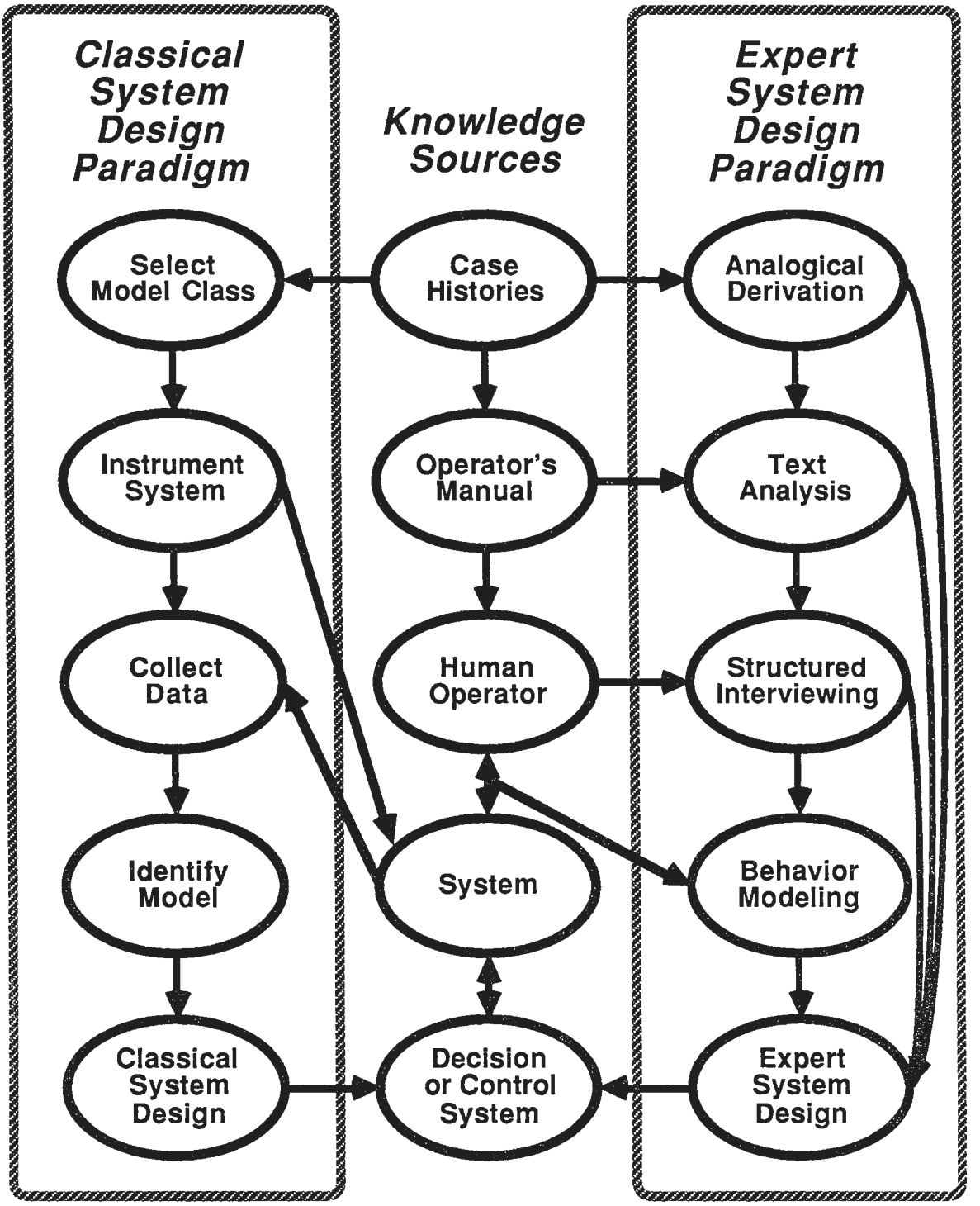


Fig.6 Classical and expert system design paradigms

The expert systems paradigm may be seen to be particularly applicable when it is not possible to model the system but there is an alternative source of data available because human operators are able to performance the decision or control task. The right hand column shows the use of knowledge sources in the expert system design paradigm:

• Structured interviewing may be used to acquire knowledge directly from the operators

- Behavior modeling may be used to identify the operators' strategies even if they are unaware of them or give incorrect ones in interviews
- Text analysis may be used with instructional material such as the operators' manuals
- Reasoning by analogy may be used based on the case histories without the data collection of the classical system design paradigm

Note that the classical and expert system design paradigms, and the various acquisition techniques for expert systems, need not be regarded as competitive. They may all be used in the system design, and it may be that a collection of weak methods is necessary because there is no overall strong method that can be applied. The resultant integration issues are analyzed in Figure 7 in terms of the distinctions involved.

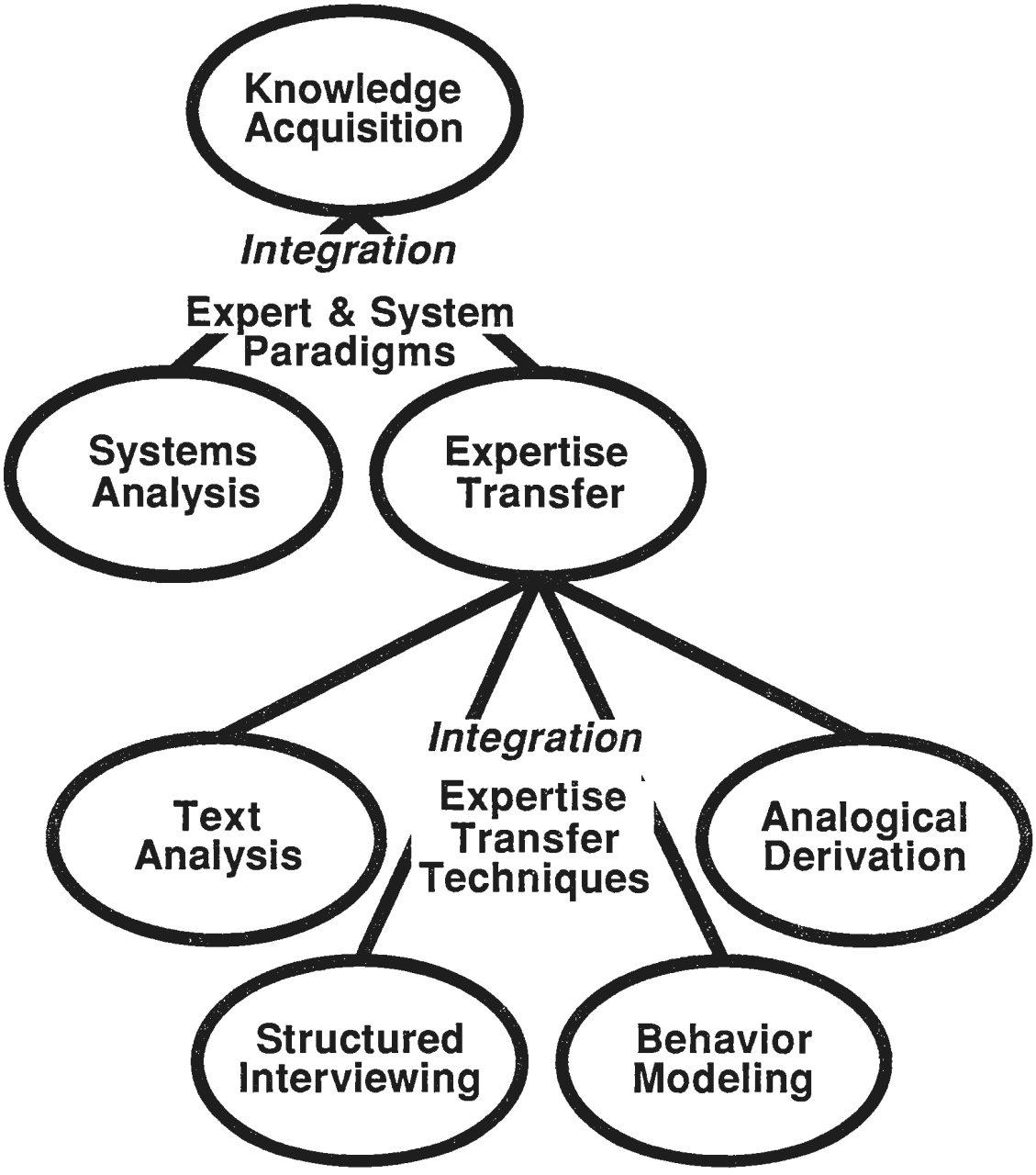


Fig.7 Major distinctions in knowledge acquisition approaches

The integration of the classical systems analysis with expertise transfer techniques is a major task. The primary problems currently are in the widely different forms of knowledge representation usually involved, and these will be noted in a later section in terms of integration of conventional and knowledge-based system components. However, at a knowledge acquisition level the problem can be seen as one of the imbalance in the types of knowledge structures studied in depth to date. There has been an imbalance favoring symbolic knowledge to redress the past emphasis on quantitative methods. This needs redressing with the knowledge structures underlying arithmetic being integrated with those underlying non-arithmetic symbol manipulation. The design requirements are:

Integration of Systems Analysis and Expertise Transfer: Provide knowledge representation that is adequate to encompass quantitative and qualitative data and reasoning.

The success of the expert systems approach should not detract from the long term significance of building deep models of the system itself. The objective performance evaluation of expert systems already discussed can provide data on deeper issues. It corresponds to learning from experience in human experts.

Separation of Systems Analysis and Expertise Transfer: Provide for system analysis of performance data from expert systems in applications to improve their performance and the depth of their models.

The integration of various techniques for expertise transfer is already a theme at the knowledge acquisition workshops. For example, case history and text analysis may be used to provide terms and concepts that aid interviewing and behavior description. The design requirements are:

Integration of Expertise Transfer Techniques: Provide intermediate knowledge representations that allow for transfer between different sources of knowledge.

The limited converse is that the different sources of knowledge, and the possibilities of contradictions between them, should be recognized.

Separation of Expertise Transfer Techniques: Keep track of the different sources of knowledge within the system and allow for contradictions in inferences from it.

INTEGRATION ISSUES OVERALL

Figure 8 combines the structures of Figures 4, 5 and 7 into an overall analysis of the distinctions underlying integration issues in knowledge acquisition for knowledge based systems. It raises some additional integration issues.

At the middle left of Figure 8 a distinction has been made between the application and explanation components of the expert system shell. This is important in acquisition because the knowledge required for explanation may be more extensive and more demanding than that required for application. Explanation facilities in most shells are very weak currently but as they develop to take into account the more advanced analyses of the nature of explanation (Hughes 1986) the requirements for explanation-oriented knowledge elicitation will increase. The design requirements are:

Integration of Application and Explanation: Explanation should be treated as a major component of application and knowledge acquisition should be targeted accordingly.

In many areas such as validation there are major differences between application and explanation that need to be taken into account.

Separation of Application and Explanation: The need for separate validation of performance in application and explanation should be taken into account.

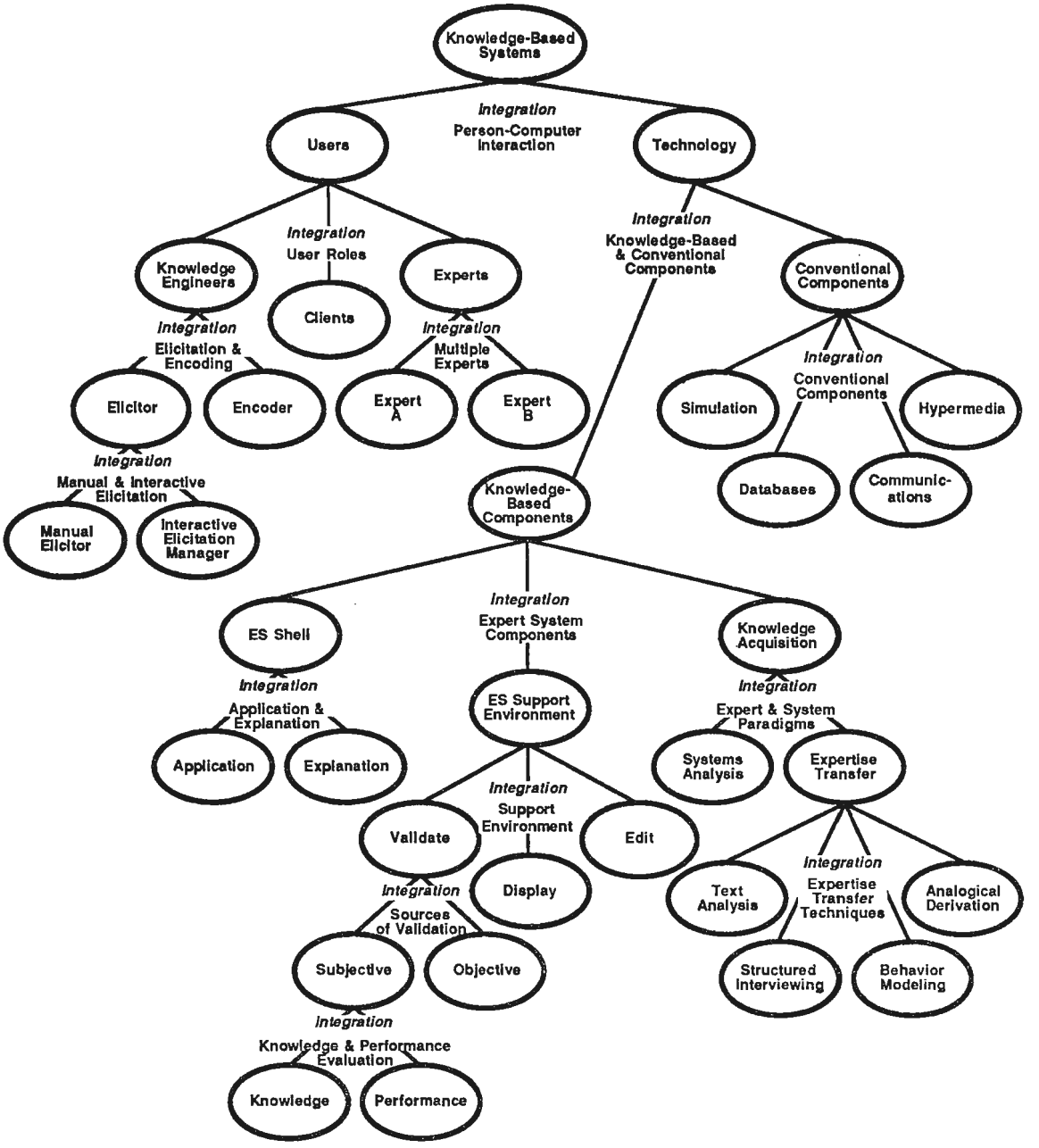


Fig.8 Overall structure of distinctions relating to knowledge acquisition integration in knowledge based systems

At the top right of Figure 8 some of the conventional components of a data processing system have been introduced and a distinction has been made between them and the knowledge-based components. The term 'knowledge' has tended to be used in a specialized way in the knowledge based system context as 'knowledge which is symbolic and computationally processible.' Much significant knowledge does not come within this narrower definition yet is still important in knowledge based systems. This leads to a need to integrate several conventional data processing technologies, for example:

- *Simulation* is important in showing the time sequences of the behavior of dynamic systems

- *Databases* provide a convenient means for storing large quantities of data with efficient retrieval, and can be used to build a variety of object hierarchies in knowledge bases from different perspectives
- *Communications* is important in allowing interaction between experts at remote sites during group elicitation processes
- *Hypermedia* provide a means for storing and retrieving knowledge even if it is not computationally processible and, in particular, provide for knowledge transfer through video material allowing mimicking at a non-verbal level

The design requirements are:

Integration of Knowledge-Based and Conventional Components: Extend the concept of a knowledge-based system to include the knowledge structures of conventional data processing.

The only danger is that some essential features of knowledge based systems are lacking in existing forms of data processing.

Separation of Knowledge-Based and Conventional Components: Extend conventional data processing to include features of knowledge based systems such as explanation.

At the top of Figure 8 the final integration is between the users and technology, that is the achievement of effective person-computer interaction. This is a major problem in all systems of the complexity of that shown in Figure 8, and is already apparent in knowledge acquisition even at the level of integration of Figure 7. The gap between functionality and usability can be very difficult to bridge, and it is easy to create multi-source, multi-format, multi-window operating environments that present great power but utter confusion to the user. The design requirements are:

Integration of Users and Technology: Provide a uniform and consistent user interface to the knowledge based system that maximizes the usability of the available functionality.

However, the interface must reflect reality and not be cosmetic.

Separation of Users and Technology: Do not use syntactic sugar and other interface tricks to give the user the impression of greater capabilities than exist in the system.

CONCLUSIONS

Tools and techniques for knowledge acquisition for knowledge based systems need to be integrated with the overall system and not treated as separate components. The variety of sources of knowledge, representations and applications within the system, and user roles, makes such integration complex. The complexity raises many issues not generally thought of as part of knowledge acquisition yet fundamentally significant in extending and integrating acquisition tools. This paper has given a preliminary analysis of the structure of knowledge-based systems as it relates to the integration of acquisition. It is intended that this structure be used as a basis for system design and it will be extended to form a 'requirements specification' for knowledge based systems integrating acquisition with other aspects of their operation.

ACKNOWLEDGEMENTS

Financial assistance for this work has been made available by the Natural Sciences and Engineering Research Council of Canada. I am grateful to many colleagues at the knowledge acquisition workshops for discussions which have highlighted many of the issues raised in this paper.

REFERENCES

Boose, J.H. & Bradshaw, J.M. (1987). Expertise transfer and complex problems: using AQUINAS as a knowledge acquisition workbench for expert systems. **International Journal of Man-Machine Studies 26**(1) (January).

Hughes, S. (1986). Question classification in rule-based systems. Bramer, M.A., Ed. **Research and Development in Expert Systems III**. pp.123-131. Cambridge: University Press.

Johnson, N. (1987). Mediating representations in knowledge elicitation. **Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge Based Systems**. pp.A2-1-10. Department of Computer Science, Reading University.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

LEARNING FROM EXAMPLES USING INFORMATION THEORY

Rodney M.F.Goodman and Padhraic Smyth
Department of Electrical Engineering, 116-81
California Institute of Technology
Pasadena, CA 91125

This paper develops and applies ideas from information theory for use in learning algorithms. We concentrate on the area of decision trees and review our recent theoretical work on the mutual information decision tree design algorithm. A new application of this algorithm is reported, where we formulated edge detection for image processing as a classification problem. We show how the algorithm learned to detect edges and yielded new domain-specific information about the problem.

Introduction

Rule induction from large data sets is currently receiving attention in the areas of machine learning and expert systems. Classifier design from labelled training samples is a problem which shares many characteristics with the rule induction problem. A recent paper by Bundy et al (1985) provides a useful discussion of how the two problems relate to each other.

The basic premise of many rule induction mechanisms, when the data is probabilistic rather than deterministic, is to induce a hierarchy or decision tree as a representation of the relationships between the attributes (evidence) and the classes (hypotheses). Hence general relationships between classes and attributes are induced or learned by the induction mechanism. Note that the terms *attributes* and *classes* occur more often in pattern recognition literature than the terms *evidence* and *hypotheses* which tend to be used in the artificial intelligence domain. For the purposes of this paper we adopt the former.

When the attribute-class relationships are probabilistic rather than deterministic, the induction mechanisms which work best appear to be those based on statistical and/or information theoretic techniques, e.g. the ID3 algorithm [Quinlan, 1983] based on information theory, the CART system [Breiman et al, 1984] for medical diagnosis, or PREDICTOR [White, 1985], a rudimentary inference tool, the latter two based on statistical techniques. Perhaps the best known inference mechanism of this type is the ID3 algorithm since it has already been implemented as part of an expert system design tool [Michie et al, 1984] which in turn has been used to design systems for weather forecasting [Riese, 1984] and fault diagnosis [Stuart et al, 1984].

In this paper we concentrate on the average mutual information heuristic for rule induction from large data sets, of which the ID3 algorithm is an example. After defining the general mutual information induction algorithm we briefly review our earlier work in this field. Primarily we have defined a general information theoretic model for the induction problem. From this model we develop several general principles. The model enables us to understand why algorithms like ID3 work so well in practice and from it we can confirm conjectures such as those of Quinlan (1986) and others. In the second part of the paper we apply the algorithm to the problem of edge detection in images. The resulting induced edge

Samples	*Attributes* A_1 A_2 ..... A_N	*Classification label*
1	0 35 7	C_4
2	1 16 4	C_2
3	1 42 9	C_{11}
⋮	⋮ ⋮ ⋮	⋮
M	0 32 8	C_4

Figure 1 : typical sample data table

detectors are both computationally faster than 'manually' designed operators and, perhaps more significantly, yield new knowledge for this problem domain. The overall conclusion reached from both theoretical and practical considerations is that the mutual information inference mechanism is indeed a powerful tool and merits further research and applications.

The mutual information decision tree algorithm

The algorithm we are about to describe is a general algorithm which derives a decision tree from a large data set of labelled samples using the average mutual information function in a 'greedy' fashion to find the most informative attributes relative to the class. Instances of this algorithm may vary considerably depending on the termination rules used, where a termination rule is essentially a mechanism for deciding that no more attributes are relevant or need not be tested. Hence Quinlan's ID3 algorithm with the Chi-square termination test [Quinlan, 1986] is a specific case, as are several others in the pattern recognition literature, [Casey and Nagy, 1984 ; Wang and Suen, 1984 ; Sethi and Sarvaryudu, 1982].

Consider that one is given a table of data as in figure 1. The table consists of M samples drawn from the population at large. Each sample is described in terms of N attribute values and a class label. For example the samples could be plants described in terms of height, number of flowers, etc. or medical case-histories in terms of presence or absence of certain symptoms. It is important to realise that the table is probabilistic rather than deterministic, i.e. there may be 'overlap' in the class-attribute conditional probability distributions.

If all known attributes are given then the algorithm will inherently select the most relevant attributes and ignore the irrelevant ones. This is an additional advantage of the mutual

```
tree( )
    begin
    if(node_list   not   empty)
       {
            node = node_list[topnode]
            if (node is a leaf)
              {
                    increment leaf_list
                    tree( )  }
            else
              {
                     A = maximum information attribute
                     for(each value of A) {
                            sort data table
                            increment node_list }
               }
         }
         else return( )
  end
```

Figure 2 : pseudo-code description of the tree classifier design algorithm.

information approach over conventional classification techniques since it yields valuable information on the relative importance of the various attributes. In applications such as medical diagnosis this can be quite useful [Breiman et al, 1984]. We assume that the sample size is sufficiently large to yield reliable estimates of the class distribution conditioned on the values of the N attributes.

Let the estimated discrete class distribution be $\underline{C}$ with K components $p(\mathrm{C} = c_i) = p_i$, $\sum_{i=1}^{K} p_i = 1$, $1 \leq i \leq K$ where c_i is the ith class and C is the class random variable. Then $H(\mathrm{C}) = \sum_{\mathrm{i}=1}^{\mathrm{K}} \mathrm{p_i} \log \frac{1}{\mathrm{p_i}}$ is the entropy of C .

There are N attributes each denoted by $\mathbf{A_i}$, $1 \leq \mathrm{i} \leq \mathrm{N}$. In turn, each attribute $\mathbf{A_i}$ has n_i values which it can assume; n_i is finite, i.e. the attribute values are quantised. $I(\mathbf{C} ; \mathbf{A_i})$ is the mutual information (as defined by Shannon) between C and $\mathbf{A_i}$.

Now we consider the algorithm itself. Essentially it just deals with one tree-node at a time. The initial node (root node) consists of the original table of data samples, i.e. unconditioned on any attribute values. Subsequent nodes result from evaluating certain attributes and obtaining sub-tables conditioned on the outcome of all prior evaluations. The algorithm continues to process nodes until there remain no candidate nodes, i.e. the tree has been grown and all leaves and internal nodes defined. The algorithm may be implemented recursively using a stack to store unprocessed nodes. New nodes which are not leaves are pushed onto the stack and 'popped' off later to either yield more new nodes (child nodes) or be declared a leaf. The algorithm is defined in pseudocode in figure 2.

Apart from the bookkeeping aspects two functions in the algorithm remain to be defined, namely the 'leaf-or-not' and 'max-information-feature' functions. It is these two functions which characterize any 'top-down' tree derivation algorithm, i.e.

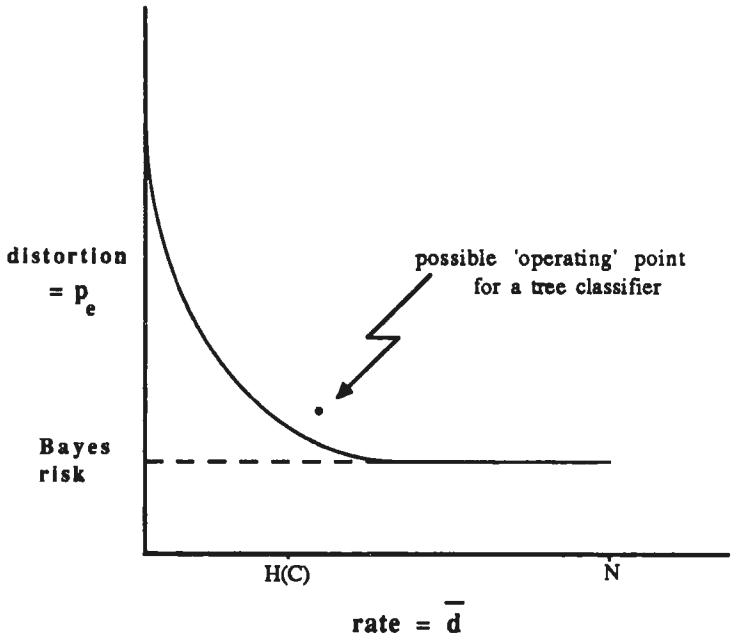


Figure 3 : general distortion-rate characteristic

(i) determine if the node is a leaf

(ii) if it is not a leaf then determine the feature or attribute which yields the most information at that node, i.e. choose $\mathbf{A_k}$ such that

$$I(\mathbf{C}\,;\mathbf{A_k}\,) \geq I(\mathbf{C}\,;\mathbf{A_i}\,) \qquad 1 \leq i \leq N,\ i \neq k$$

or equivalently,

$$H(\mathbf{C}\,|\mathbf{A_k}\,) \leq H(\mathbf{C}\,|\mathbf{A_i}\,) \qquad 1 \leq i \leq N,\ i \neq k.$$

In the next section we summarise our recent work on this algorithm.

Theoretical results for the algorithm

Prior work on this algorithm has been applications motivated, as for example the various applications reported in the field of optical character recognition [Casey and Nagy, 1984 ; Sethi and Sarvarayudu, 1982]. The work of Quinlan (1986, 1983) has suggested that the algorithm can be used in the more general domain of rule induction from examples. Hence our earlier work [Goodman and Smyth, 1987, 1986] in this field was directed towards a better understanding of the algorithm with a view to applications in rule induction for expert systems. In particular we examined such isues as average path length/misclassification rate trade-offs, the effect of increasing the noise on the termination rules and the overall effectiveness of the mutual information heuristic. In this section we present a brief summary of these results.

We begin by modelling the problem in a communication theory setting. This communication theory model was recently suggested by both ourselves and Chou and Gray (1986) who proposed an equivalent rate-distortion model. Essentially the classification problem is equivalent to a communications problem where one is communicating through a noisy channel at a variable rate. The noisy channel represents the overlap in the class-attribute conditional probability densities, and is equivalent to the noise which Quinlan (1986) discussed. The general idea is to transmit the most informative bits (or attributes) to give the decoder (decision tree scheme) the best chance of decoding the message (class). Hence we can immediately grasp the fundamental distortion-rate trade-off, more bits meaning less distortion.

This trade-off is formalised in communication theory by the rate-distortion curve for a given noisy channel which provides a fundamental lower bound on the achievable distortion for a given rate. By modelling the induction problem in this setting the rate becomes the average tree depth, $\bar{d}$, while the distortion is the average probability of misclassification, P_e, in using the tree. Figure 3 shows a typical distortion-rate curve. The horizontal assymptote represents the minimum achievable misclassification rate, P_{min}, for a given classification problem. In pattern recognition literature this is called Bayes' risk and is a result of the inherent noise or overlap in a given problem. Hence the distortion-rate curve provides a lower bound for what might be termed the 'operating point' of the tree classifier. A good algorithm will induce classification rules which have lower distortion, P_e, as the rate, $\bar{d}$, increases. We have shown in previous work [Goodman and Smyth, 1987] that for certain termination rules P_e is indeed bounded above by a function which decreases as $\bar{d}$ increases.

We can also use the distortion-rate model to confirm and explain an earlier conjecture [Quinlan, 1986], namely that it is better to use noisy data than 'non-noisy' good data in the design phase, if the induced classification rules are to be used on noisy data. Figure 4 shows clearly that this is true. Consider that we wish to design a tree to achieve some specified P_e. Using 'non-noisy' data is like designing the classifier with an artificially low distortion-rate bound ; $\bar{d}$ is fixed at an artificially low value of P_e. In using the tree P_e will be larger, perhaps much larger, than the specified P_e. However if noisy data had been used, the designed value of P_e and the true value would be the same, with perhaps only a slight increase in $\bar{d}$.

Continuing the communications analogy we have also shown that the top-down mutual information algorithm is equivalent in a certain sense to Shannon-Fano prefix-coding [Fano, 1961]. Choosing the attribute which yields the most information is equivalent to choosing the attribute which most closely partitions the classes into subsets of equal probability. The Shannon-Fano prefix-coding scheme is known to be near-optimal compared to the optimal scheme by Huffman [Huffman, 1952]. This equivalence to prefix-coding is a very positive result with respect to the mutual information induction heuristic. We have established upper and lower bounds on $\bar{d}$ (in terms of the entropy of the class distribution $H(\mathrm{C})$ and the noise level) using the coding ideas. The tightness of the bounds obtained [Goodman and Smyth, 1987] are further proof that the algorithm is near-optimal. For example with no noise, and with some constraints on the class probability distribution, we get

$$H(\mathrm{C}) \leq \bar{\mathrm{d}} \leq 1.09\mathrm{H}(\mathrm{C})$$

The rate-distortion model and the prefix coding equivalence provided the basis for some new results regarding termination rules. From figure 3 we see that the form of the termination rule will critically influence the position of the 'operating point' of the classifier.

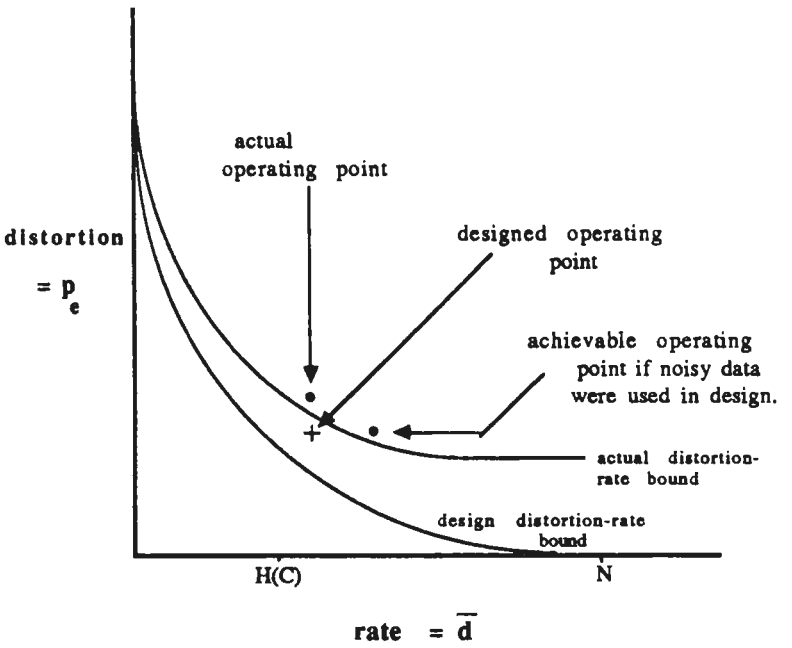


Figure 4 : how 'good' data can lead to the wrong result

We also showed [Goodman and Smyth, 1987] that the information gained at any node is simply the entropy of the branch probabilities less a noise term. The size of the noise term limits the available information. Prior work in this area had indicated that rules using thresholds did not perform well in practice [Breiman et al, 1984 ; Sethi and Sarvarayudu, 1982]. In terms of our model it is clear why this happens due to the basic phenomenon of limited information in the presence of noise. Termination rules based on statistical tests such as the Chi-square test [Quinlan, 1986] are an improvement but may not be sufficient due to limitations on the sample size, i.e. the rule is most critically needed at nodes deep in the tree - but at such nodes the number of samples may be very small, precluding the use of the Chi-square test. Consequently we have proposed a new termination rule called the Delta-entropy rule which basically compares the normalised entropies of the class distribution at a node, with and without p_{max}, and decides to terminate or not based on whichever is larger [Goodman and Smyth, 1986]. The rule has been found to work well in practice and is robust in the presence of noise.

In summary for this section, the main findings of our earlier work has been

(1) to confirm that the mutual information induction algorithms (such as ID3) are indeed near optimal,

(2) provide quantitative bounds on the performance of such algorithms, and

(3) gain a general understanding of such issues as termination rules and choice of design data, and consequently develop a new robust termination procedure.

Having gained considerable *theoretical* insight into the performance of the mutual information mechanism we turned our attention to a *practical* application. The remainder of this paper reports on the results of our practical work.

An application to edge detection

Edge detection remains one of the fundamental issues in the field of computer vision. For the purposes of this paper we will use the description 'local discontinuity in image luminance' to define an edge. An image is sampled and stored as a finite set of discrete grey values. It is not surprising then that most edge detection schemes are essentially digital filters through which the image is passed or convolved. The length of the filter is defined by the size or extent of the window operator used to implement the filter. The filters tend to be non-causal and non-recursive, i.e. finite impulse response. Despite the fact that the window sizes of typical edge-detector filters are relatively small in comparison with the size of the image itself, two-dimensional convolution is still an expensive operation in computational terms. Recent advances in edge detection theory [Canny, 1983 ; Marr and Hildreth, 1980] have led to the derivation of filters which are optimal under certain criteria. Unfortunately the window sizes of these filters are large enough so that for the present they are not realisable for many applications such as real-time automated inspection. Instead the smaller window size operators such as the Sobel [Duda and Hart, 1973] continue to be used in the applications environment. However even for small size windows (e.g. 3 x 3) the bottleneck remains that two-dimensional convolution is computationally slow. There is considerable motivation then to investigate any techniques which can reduce the fundamental computation requirements in edge detection.

We introduce here a new approach (from a rule induction viewpoint) to edge filtering with a resulting significant decrease in computation cost. Essentially we formulate the edge detection problem as a classification problem where pixels are to be classified as either 'edge' or 'non-edge'. Because the mutual information induction algorithm yields a rule *hierarchy*, it is ideally suited to the problem of deriving faster operators.

After defining a general procedure for designing hierarchical edge operators we proceed to derive such operators on particular training images and evaluate their performance on these and other images. For comparison purposes we use two other operators namely the afore-mentioned Sobel operator and the recently introduced dispersion operator [Pitas and Venetsanopoulos, 1986] based on order statistics. The reasons for choosing precisely these operators are explained later but it is sufficient to note here that of the local (3 x 3 window) edge detectors the Sobel is adjudged to perform best [Abdou and Pratt, 1979] while the dispersion operator is in principle the fastest. The next section gives a brief description of these operators. It is important to note that the 3 x 3 size window was chosen for convenience only and the method outlined is perfectly general for any arbitrary window sizes.

The Sobel and Dispersion edge operators

For evaluation purposes we have chosen to use the Sobel operator and the less well known dispersion operator to compare with the hierarchical approach. The pixel notation is given in figure 5 and the two masks for the Sobel operator are defined in figure 6.

The procedure is to obtain 2 edge-enhanced images by convolving these masks with the original image - essentially one mask enhances vertically oriented edges while the other enhances horizontally oriented edges. If we define EI(i,j) to be the grey-scale value of the enhanced image at pixel (i,j) then,

$$\mathrm{EI}_1(\mathrm{i,j}) \;=\; \Big|\sum_{k=1}^{3}(x_k - x_{k+6})\Big|$$

1	2	3
4	5	6
7	8	9

Figure 5 : pixel notation

$$\mathrm{EI}_2(\mathrm{i,j}) \;=\; |\sum_{k=1}^{3}(x_{3k-2} - x_{3k})|$$

(where x_k is the grey-scale value of the kth pixel (figure 5), so that

$$\mathrm{EI}(\mathrm{i,j}) \;=\; \mathrm{EI}_1(\mathrm{i,j}) + \mathrm{EI}_2(\mathrm{i,j})$$

A suitable threshold t is determined and a pixel is classified as an edge if

$$\mathrm{EI}(\mathrm{i,j}) \;>\; \mathrm{t}$$

otherwise it is classified as a non-edge.

The dispersion operator was recently introduced by Pitas and Venetsanopoulos (1986) and has the advantage of being in principle computationally faster than any other known 3×3 operator. It is based on order statistics. To obtain the order statistics it is first necessary to sort the N pixels in the window in order of increasing magnitude so that

$$x_{(1)} \;\leq\; x_{(2)} \;\leq, \ldots\ldots, \leq\; x_{(N)}$$

where $N = 9$ for a 3×3 window. The range of the random variables $x_1, \ldots, x_N$ is defined as

$$w_{(1)} \;=\; x_{(N)} - x_{(1)}$$

Quasi-ranges are defined as

$$w_{(i)} \;=\; x_{(N+1-i)} - x_{(i)} \qquad , \qquad 2 \leq i \leq \lfloor \frac{N}{2} \rfloor$$

One can define the range and quasi-ranges as primitive operators; they give an indication of the grey-scale variation within the local area of the pixel and hence indicate the possible presence of an edge. However due to the absence of spatial information they are susceptible to noise and in particular impulse noise. The dispersion operator was defined as an average over the quasi-ranges to combat the effect of noise, i.e.

$$W = \sum_{i=1}^{\lfloor \frac{N}{2} \rfloor} w_i$$

1	2	1
0	0	0
-1	-2	-1

1	0	-1
2	0	-2
1	0	-1

Figure 6 : the vertical and horizontal Sobel edge masks

The W value for each pixel (i, j) is thresholded to determine if or not it contains an edge. It can be shown [Pitas and Venetsanopoulos, 1986] that this operator performs almost as well as the Sobel operator over a variety of comparison measures and can be implemented much faster.

Edge detection as a classification problem

The idea of pixel-based classification into 'edge' and 'non-edge' classes is easy to visualise. As in any classification problem the classes are described in terms of attributes. A classifier is designed by taking a sufficiently large sample of pixels in order to estimate the conditional probabilities of the classes given the various attributes, e.g. the probability that the class is 'edge' given that the ith attribute has value j. From the set of training samples relating attributes and classes we infer some general rules for classifying pixels based on their attribute description, using the mutual information induction algorithm.

First however the problem of attribute selection must be addressed. Remember that our ultimate goal is to reduce the computation cost in defining a local edge detector. Consequently, in order to compete with for example the Sobel operator, whatever attributes we choose must be computationally simple to implement - the term 'primitive' provides a convenient description. Window operators can be considered as 'composite' binary attributes in the sense that they are composed of combinations of primitive attributes. They are

binary-valued because the pixels are essentially classified as 'edge' or 'non-edge' according to whether the composite attribute is greater than or less than a certain threshold. What might be the appropriate primitive attributes? We claim that for edge detection the difference operator is the most important type of primitive operator. The rationale for this claim can be stated in many different ways but in simple terms the presence or abscence of an edge will lead to larger or smaller difference values - although the converse may not be true due to the presence of noise. Primitive attributes can be defined by supplying pairs of arguments to the difference operator. Consider the standard 3 x 3 window as shown in figure 5 with associated notation. We could for example define the following primitive attributes ,

$$\begin{aligned} \text{attribute 1} &:= x_1 - x_9 \\ \text{attribute 2} &:= x_4 - x_6 \\ \text{attribute 3} &:= \max_i\{x_i\} - \min_i\{x_i\} \end{aligned}$$

where the i ranges over the window as described in figure 5 and x_i is the grey level at the ith pixel. It is easy to see that the Sobel operator can be defined in terms of primitive attributes like the ones above. We use F_i to represent the ith attribute and the random variable f_i to denote the value of F_i. f_i is a discrete random variable, which for the difference-type operators we are discussing here, is restricted in range to the maximum grey level of the image.

The problem of course is that there is an extremely large set of potential pairwise attributes to choose from (e.g. we are not necessarily constrained to using pixels within a 3×3 window) - which ones should be chosen? As in any classification-type problem attribute selection is almost an art form (however one of the advantages of the mutual information algorithm is to delete irrelevant attributes from consideration, making the attribute selection process easier). For the purposes of this paper we have restricted our attributes to be the primitive operators which make up the Sobel and dispersion operators - this facilitates comparison on both a performance and computational level. It is important to realise however, that throughout the rest of the discussion, a primitive attribute can be thought of as *any* primitive pairwise operator defined on pixels. We note again that the composite operators or attributes are essentially binary-valued, i.e. greater than or less some threshold. In computation terms this amounts to a simple comparison. Hence we must also implement the primitive attributes in this manner (i.e. with thresholds) if we wish the hierarchical operator to be competitive in computational terms. The old problem of threshold selection once again rears its ugly head. There are a variety of schemes [e.g., Abdou and Pratt, 1979], for calculating threshold parameters for edge detectors subject to some optimisation criterion. However we cannot strictly apply any of those techniques for *hierarchical* edge detection, the reasoning being as follows. Consider the conditional probability of the class 'edge' being present given that some binary primitive attribute, F_i, is greater than some threshold, i.e. $p(edge|f_i > t)$, where f_i is the random variable denoting the value of the attribute F_i, and t is the threshold. Since the pixels are correlated it is obvious that the various class and attribute probabilities are not independent, i.e. if attribute x has some value then the probability that 'there is an edge given attribute y greater than some other value' must be conditioned on attribute x. So for example the Bayesian minimum error threshold t_{opt} is chosen to minimise the quantity

$$\begin{aligned} p(error) &= p(edge, f_i \leq t) + p(non\ edge, f_i > t) \\ &= p(edge|f_i \leq t) \times p(f_i \leq t) + p(non\ edge|f_i > t) \times p(f_i > t) \end{aligned}$$

In a hierarchical classifier the above probabilities must be conditioned on all the previously evaluated attributes and so t_{opt} depends on these attributes. Hence one must calculate all the possible $N(N-1)$ optimal thresholds before designing the classifier (N is the number of attributes). This is not practical given the inherent difficulties in estimating the probabilities in the equation for $p(error)$.

Instead we propose an approach which circumvents the problem by using the tree derivation algorithm. As we have seen the algorithm works by choosing the best attribute (in an information-theoretic sense) conditioned on whatever attributes have already been chosen. At any given node in the tree the best threshold is found for each attribute. This is achieved by use of the Kolgomorov-Smirnov test, which has certain desirable properties for this problem [Friedman, 1977]. Given the thresholds, the attribute which then yields the most information about the class variable is selected at that node.

Having defined the appropriate attributes, the next step is to obtain a set of labelled or classified samples, i.e. training samples. Define a 'typical image' to be one that contains typical edge content, luminance and noise of those images to be encountered in practice. If the variation in application images is significant then from a practical point of view it may be necessary to combine segments of different images in order to synthesize a training image ; since the individual samples are pixels then it quite easy to obtain a large sample so that the only constraints imposed may be by the computing facilities available to run the classifier design algorithm, e.g. memory constraints. The typical image is then classified via standard edge-detection techniques using the best edge-detection algorithm available and the table of training samples is obtained. Each datum corresponds to a pixel described by the chosen primitive attributes evaluated at that pixel and tagged with an 'edge' or 'non-edge' label.

We can now begin to appreciate the potential advantages of the hierarchical operator. If for example the typical image contains low edge content, as indeed is the case with most images, then the optimal hierarchical operator should first try to classify each pixel as 'non-edge' using some primitive attribute if possible and for those pixels which are not immediately so classified apply other attributes to determine if an edge is present. Since many images have at least 90% of their pixels as 'non-edge' the computational advantages are obvious.

For the hierarchical operator to be used for comparison purposes we chose to define the primitive attributes as the range and the quasi-ranges of the order statistics model, and the corner differences and main axes differences of the Sobel operator as defined in figure 7. The rationale for choosing the latter 4 attributes (as defined in figure 7) was to include some primitive attributes which had spatial dependence and could indicate edges in various orientations. The attributes chosen may not be the set of optimal primitive attributes but rather are a set of easily computable operations which are easy to interpret. However it is important to remember that any redundant attributes which might be chosen, or equivalently attributes which yield no relevant information, will by definition be suppressed by the tree derivation algorithm. The goal here is not to necessarily to derive *optimal* hierarchical operators but rather demonstrate the feasiblity of hierarchical operator design and investigate the resulting performance.

Having defined the attributes the next step is to run the algorithm given in section 2 on the training samples. We are interested in three aspects of the results, namely

(1) The performance of the induced edge operator

(2) The possibility of inducing new 'knowledge' about edge detection in general

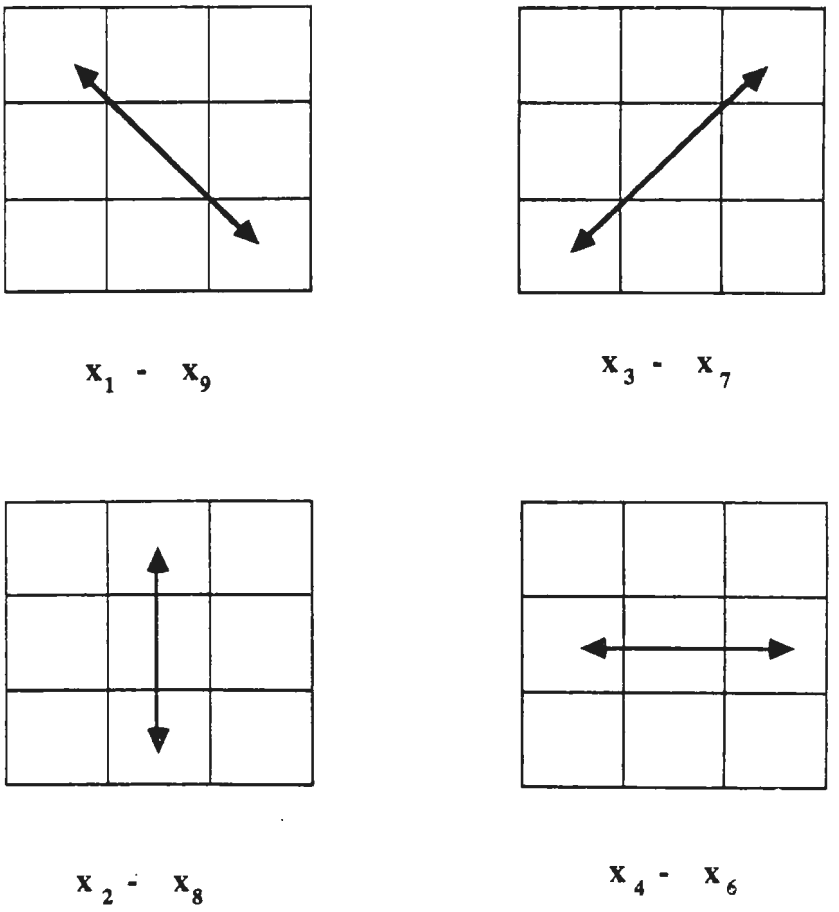


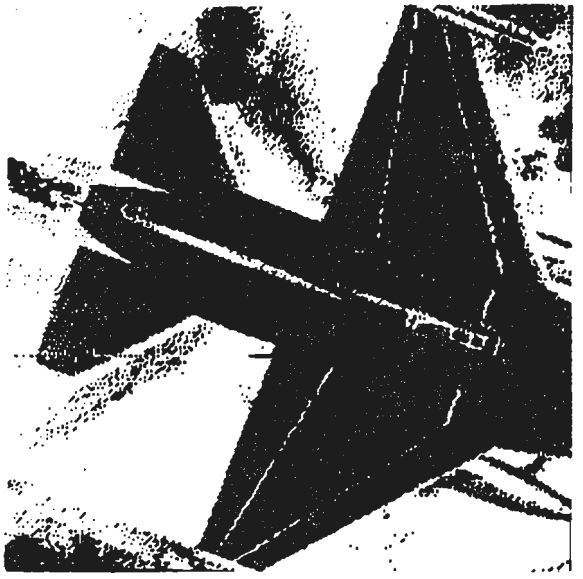
Figure 7 : spatial primitive features

(3) Overall implications regarding the mutual information induction algorithm and its application.

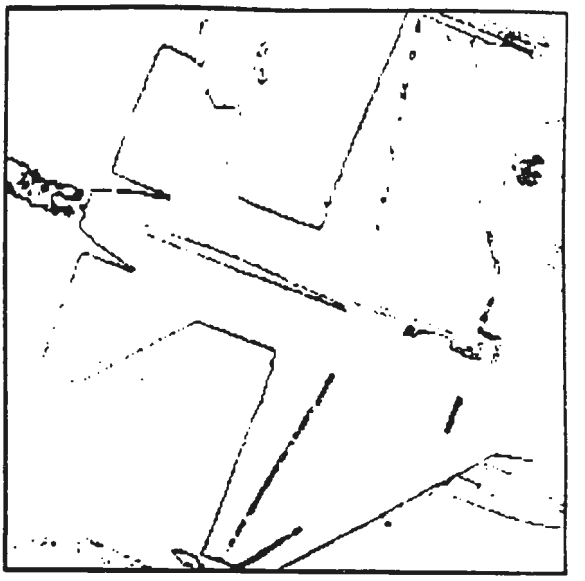
Experimental results

In this section we compare the performance of the Sobel, dispersion and hierarchical operators on 2 different images. The images were generated via a scanning device using 32 grey levels. Each image is 256 pixels by 256 pixels.

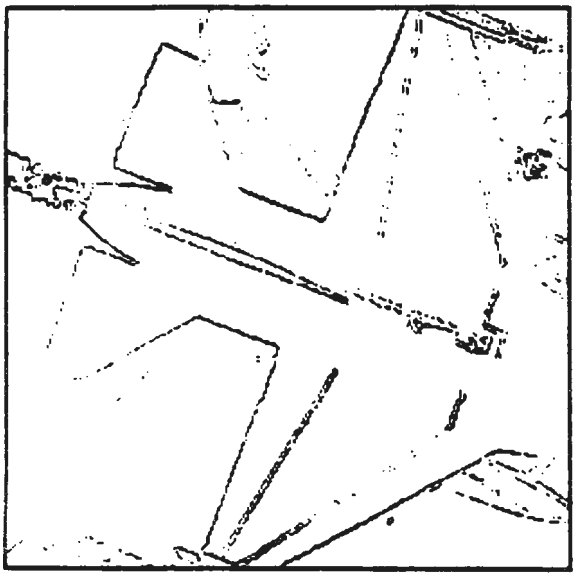
Before considering the results it is worth mentioning some details. The operator used to classify the samples was the Sobel operator since we found that it yielded slightly higher quality edge maps than the dispersion operator. The threshold t for both operators was chosen heuristically based on visual evaluation of the effects on the edge maps of varying t. Because of memory limitations in our software the entire image could not be used for training purposes. Consequently 80 pixel by 80 pixel size sub-images were used as training images. Provided the sub-images were reasonably typical of the image as a whole it made



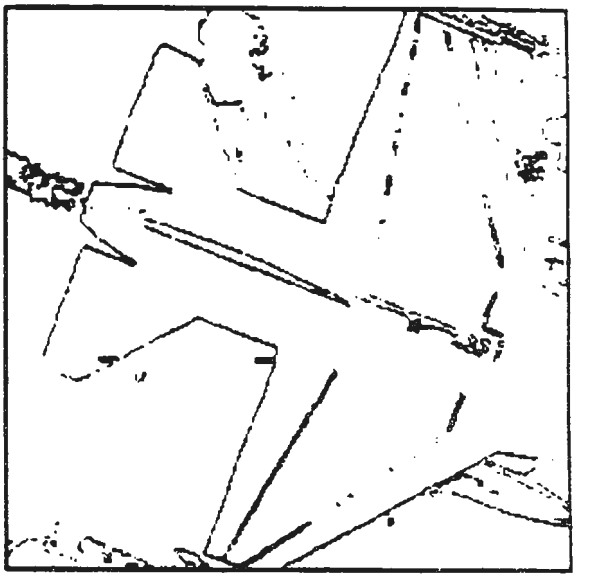
(a) half-toned original



(b) Dispersion



(c) Sobel



(d) Hierarchical

Figure 8 : Edge maps for 'airplane' image

little difference in the experiments where the sub-image was located within the overall image. In each of figures 8 and 9, the sub-figure(a) is a half-toned output on a printer of the original grey-scale image.

The first example we consider is the airplane image shown in figure 8(a). The output of the Sobel operator in figure 8(c) is clearly better than that of the dispersion operator in



(a) half-toned original



(b) Dispersion



(c) Sobel



(d) Hierarchical

Figure 9 : Edge maps for 'bin of tools' image

8(b). The hierarchical output in 8(d) is also better than the dispersion. But it is difficult to distinguish whether the Sobel or hierarchical operator is better. While the Sobel picks up finer detail the hierarchical operator contains more edge information, particularly that of the airplane itself. The lack of fine detail in 8(d) can be explained by the actual tree structure of the operator as shown in figure 10. It uses mainly as primitive components the

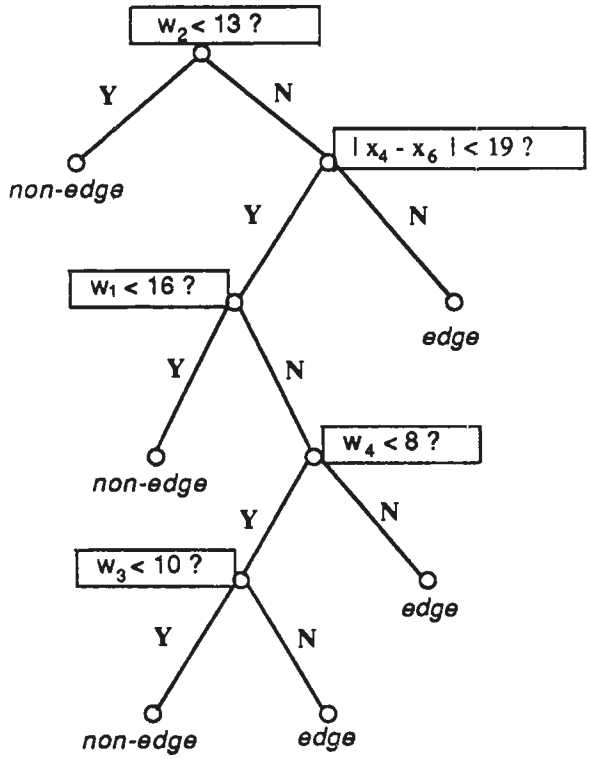


Figure 10 : hierarchical operator for airplane image

quasi-ranges which as we can see in 8(b) tends to merge edges which are close together (this difference can be seen by considering the effect of passing the 2 operators over alternating black and white pixel-wide vertical lines). The average number of additions/subtractions performed by the tree is much less that of the other 2 methods, i.e. only 1.34 as compared to 13 for the Sobel and 7 for the dispersion.

Figure 9 contains equivalent information to figure 8 but for another image - a bin of tools. Interestingly enough the hierarchical operator derived for this image was quite similar to that derived from the airplane image. All 3 operators perform similarly here with perhaps the order of merit being dispersion, Sobel, hierarchical. The difference in performance is slight. The image itself does not have clearly defined edges, hence the noisiness towards the top of all three edge maps. For this image we see that the hierarchical operator can only resolve edges as well as its primitive attributes will allow - better performance could be achieved by using a higher quality classifier than the Sobel operator and defining more primitive attributes.

Now consider the relative *computational* performance of the three operators. The results are tabulated in figure 11. Since the hierarchical operators for both images were very similar, the figures in the hierarchical column are an average over both. The conditional evaluations are tests of the form 'is $x > y$?' and do not include the computation of x if x is a compound expression - such computations are included under 'additions'. We have used the minimum delay of 8 comparisons for a parallel sorting structure [Knuth,

	Sobel	Dispersion	Hierarchical
Additions	*13*	*7*	*1.34*
Multiplications	*0*	*0*	*0*
Absolute value operations	*2*	*0*	*0.46*
Conditional evaluations	*1*	*1*	*1.34*
Sorting comparisons ;			
Parallel	*0*	*8*	*8*
Serial	*0*	*19*	*15.8*

Figure 11 : comparison of edge detectors in terms of computation

1973]. For sorting in serial form the number in the dispersion column corresponds to the minimum number of comparisons required to sort 9 numbers [Knuth, 1973]. Because we do not need to sort all the numbers each time the hierarchical operator is used, the figures in the hierarchical column are the experimentally obtained average number of comparisons required.

Any comparison in this manner is obviously dependent on the method of implementation, e.g. hardware or software, serial or parallel, relative speed of the various primitive operations. Nevertheless, independent of implementation, the hierarchical operator will always be considerably faster than the dispersion operator particularly for images of low edge content, which is the usual case. Comparing the Sobel and hierarchical operators will depend on the implementation and especially on the relative speed of comparisons and additions. If one assumes (as in, Pitas and Vetsanopoulos, 1986) that one were to implement the sorting operation using a parallel VLSI sorting network then the hierarchical operator would indeed be much faster. As a matter of interest, using programs written in C, to classify all pixels in a 256 by 256 size image took 30.8 seconds with the dispersion operator, 23.7 seconds with the Sobel, and 16.4 seconds with the hierarchical. It was apparent that both the dispersion and hierarchical operators spent most of the time sorting the pixel values, so that one might expect a parallel implementation to run very quickly indeed.

Concluding remarks

The question we addressed in this paper was basically the following : given that induction

algorithms which use mutual information (such as ID3) are intuitively appealing, can we then prove their utility both theoretically and practically ? (The answer to the latter part of the question has already been proven true to some extent elsewhere [Michie et al, 1984 ; Wang and Suen, 1984 ; Casey and Nagy, 1984 ; Quinlan, 1983 ; Sethi and Sarvaryudu, 1982].) The results we have outlined in section 3 of this paper certainly provide a sound theoretical basis for the algorithm and justify its use. From a practical viewpoint we have also met with success, i.e. the edge detection experiment. It is important to realise the general implications we can draw from our practical results.

(1) The algorithm yields efficient and accurate 'classifiers' or 'rule-hierarchies'.

(2) Attribute selection is an important aspect of experiment design. Indeed it is about the only part of the design procedure which may require domain dependent expert knowledge. However because the mutual information approach ignores irrelevant attributes, then one can over-specify the number of attributes. Hence, as in this problem, image-processing 'novices' such as ourselves can achieve relatively 'expert' results, i.e. see figures 8 and 9.

(3) We noted that in all the hierarchical operators derived by the algorithm (for the 2 examples given and several other images not shown in this paper), the attribute $w_{(2)}$ was always at the root node, i.e. $w_{(2)}$ was the most informative attribute for every image we used. In addition the spatial attributes rarely appeared in any of the operators. Hence the conjecture is that order statistics are more useful for edge detection than spatial operators and for 3×3 windows $w_{(2)}$ is the single best statistic. This is an example of new domain-dependent knowledge resulting from the induction algorithm.

We conclude that the mutual information algorithm is well-founded and has excellent practical potential as an induction tool. The immediate goal in our research is to generalise the algorithm for applications in expert systems design. At present the induced tree structure may not be flexible enough in itself for an expert system type of environment. For instance if data is unavailable at run time the hierarchy should be such as to allow for this (White also mentions this idea [White, 1985]). The existing algorithm can easily accomodate this in principle simply by deleting the appropriate attribute data at each node as if it did not exist and so obtaining an extra 'don't know' branch. However there are practical problems associated with this approach, such as exponential order tree growth, which must be solved.

Hence the current challenge as we see it is to further develop this well-founded and practically useful algorithm, which has its roots in classical pattern recognition, and move it into the more general domain of knowledge-based systems.

References

Abdou, I.E. and Pratt, W.K., 'Quantitative design and evaluation of enhancement/ thresholding edge detectors', *Proc. IEEE* , vol.67, no.5, pp.753-763, May 1979.

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

Bundy, A., Silver, B. and Plummer, D., 'An analytical comparison of some rule-learning programs', *Artificial Intelligence* ,27, 1985, pp.137-181.

Canny, J.F., 'Finding edges and lines in images', *M.I.T. Artificial Intelligence Lab., Rep. AI-TR-720*, June 1983.

Casey, R.G. and Nagy, G., 'Decision tree design using a probabilistic model', *IEEE Trans. Inform. Theory*, vol.IT-30, pp.93-99, Jan. 1984.

Chou, P.A. and Gray, R.M., 'On decision trees for pattern recognition', presented at the 1986 IEEE Int. Symp. Inform. Theory, Ann Arbor, Michigan, Oct. 1986.

Duda, R.O. and Hart, P.E., *Pattern classification and scene analysis* , New York : Wiley, p.271, 1973.

Fano, R.M., *Transmission of information*, Cambridge,MA. : MIT press, 1961, p.187 .

Friedman, J.H., 'A recursive partitioning decision rule for non-parametric classification', *IEEE Trans. on Computers*, vol.C-26, 1977, pp.404-408.

Goodman, R.M.F. and Smyth, P., 'An information theoretic approach to decision tree design', presented at the 1986 IEEE Int. Symp. Inform. Theory, Ann Arbor, Michigan, Oct.1986.

Goodman, R.M.F. and Smyth, P., 'Decision tree design from a communication theory standpoint', submitted for publication to *IEEE Trans. Inform. Theory.*

Huffman, D.A., 'A method for the construction of minimum redundancy codes', *Proc. IRE*, vol.40, pp.1098-1101, Sept. 1952.

Knuth, D.E., *The art of computer programming*, vol.3, Reading MA. : Addison-Wesley, 1973, p.231.

Marr, D. and Hildreth, E., 'Theory of edge detection', *Proc. Royal Soc. London B*, vol.207, pp.187-217, 1980.

Michie, D., Muggleton, S., Riese, C. and Zubrick, S., 'RuleMaster - a second generation knowledge engineering facility', *Proceedings of the First Conference on Artificial Intelligence Applications*, Denver CO, December 1984.

Pitas, I. and Venetsanopoulos, A.N., 'Edge detectors based on nonlinear filters', *IEEE Trans. Pattern Anal. Machine Intell.*, vol.PAMI-8, 1986, no.4, pp.538-550,

Quinlan, J.R., 'Learning efficient classification procedures and their application to chess endgames', *Machine learning : an artificial intelligence approach*, R.S.Michalski, J.G.Carbonell and T.M.Mitchell (eds.), Palo Alto, CA. : Tioga, 1983.

Quinlan, J.R., 'The effect of noise on concept learning', *Machine learning volume 2*, R.S.Michalski, J.G.Carbonell and T.M.Mitchell (eds.), Morgan Kaufmann Publishers, 1986, pp.149-166.

Riese, C., 'RuleMaster : control strategies', *Radian Technical Report RI-RS-00296*, Radian Corporation, Austin TX, December 1984.

Sethi, I.K. and Sarvarayudu, G.P.R., 'Hierarchical classifier design using mutual information', *IEEE Trans. Pattern Anal. Machine Intell.*, vol.PAMI-4, pp.441-445, July 1982.

Stuart, J.D., Pardue, S.D., Carr, L.S., Feldcamp, D.A., 'TITAN : an expert system to assist in troubleshooting the Texas Instruments 990 minicomputer system', *Radian Technical Report ST-RS-00974*, Radian Corporation, Austin TX, December 1984.

Wang, Q.R. and Suen, C.Y., 'Analysis and design of a decision tree based on entropy reduction and its application to large character set recognition', *IEEE Trans. Pattern Anal. Machine Intell.*, vol.PAMI-4, pp.406-417, July 1984.

White, A.P., 'PREDICTOR : an alternative approach to uncertain inference in expert systems', *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles CA, 1985, pp.328-330.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

Acquiring Strategic Knowledge from Experts

Thomas Gruber
Experimental Knowledge Systems Laboratory
Computer and Information Science
University of Massachusetts
Amherst, MA 01003

Abstract

This paper presents an approach to the problem of acquiring strategic knowledge from experts. Strategic knowledge is used to decide what course of action to take, when there are conflicting goals to achieve and the effects of actions are not known in advance. We show how strategic knowledge challenges the current approaches to knowledge acquisition: knowledge engineering, interactive tools for experts, and machine learning. We present a knowledge acquisition methodology embodied by an interactive tool that draws from each approach, automating much of what is currently performed by knowledge engineers, and synthesizing interactive and automatic learning techniques. The technique for eliciting strategic knowledge from experts and transforming it into an executable form addresses the technical problems of operationalization, encoding examples, biasing generalization, and the new terms problem.

1 Introduction

This paper presents an approach to the problem of acquiring strategic knowledge from experts. We discuss the motivation for acquiring strategic knowledge, describe how it challenges the current approaches to knowledge acquisition, and present a methodology for acquiring it. The technique is embodied in an interactive knowledge acquisition tool for experts.

We define *strategic knowledge* as knowledge used to decide what course of action to take when there are conflicting goals to achieve and the precise effects of actions cannot be known in advance. By actions we mean events chosen for their effects on the world, rather than just the

This research was sponsored by DARPA-RADC contract F30602-85-C0014 and ONR University Research Initiative grant N00014-86-K-1764.

internal state of the problem solver. For example, driving through a city is a task requiring strategic knowledge. The driver chooses actions, such as "take the highway" or "turn right here," to balance goals such as minimizing travel time, avoiding unpleasant or dangerous areas, and keeping the car working. The heuristic "when in heavy traffic on a road with traffic lights and two lanes in each direction, stay out of the left hand lane" is strategic knowledge for coping with left-turn snarls, which can thwart the travel time goal. Strategic knowledge is distinguished from the substantive knowledge of a domain, such as the rule "right hand turns are permitted at red lights unless prohibited by a sign." Strategic knowledge is about how to play, rather than the rules of the game.

Strategic knowledge is intrinsic to tasks in which managing the *process* of problem solving, choosing among possible actions, is part of the expertise. For example, although diagnosis is usually treated in AI as an inferential problem (i.e., classification), physicians make strategic decisions in the process of choosing diagnostic tests and trial therapeutic actions; they consider the effect of their actions on multiple goals such as patient safety and comfort, economic cost, protection against missing a dangerous condition, and, of course, making the correct diagnosis. To build systems with strategic expertise we need a technology for acquiring strategic knowledge from experts.

A fundamental problem in knowledge acquisition is the *representation mismatch* between the form of the knowledge available in the world (e.g., as expressed by experts) and the form of knowledge needed to drive a computer program (e.g., as data structures and algorithms) (Buchanan, Barstow, Bechtel, Bennett, Clancey, Kulikowski, Mitchell, and Waterman, 1983). Representation mismatch is especially acute for strategic knowledge. Experts can *demonstrate* their expertise with cases of problem solving in which they make strategic choices, and can give ad hoc justifications for their decisions. But strategy is only implicit in the experts' demonstrations, and not couched in operational terms. It is difficult for experts to explicitly *formulate* their strategic knowledge in a representation that satisfies the requirements for an expert system: the knowledge must be operational (capable of being executed to achieve the desired performance) and general (applying to cases not yet seen). The problem is that we have no way of modeling strategic knowledge in an expert system that is natural for experts.

The approach we have chosen is to provide two ways to express strategic knowledge: one that is accessible to experts, and the other that can be used to control an expert system. The technical objectives are to design an acquirable representation for the expert and an operational representation for the system, and to build a tool that can assist in transforming strategic knowledge from the acquirable representation to the operational representation. We will describe a methodology for achieving these objectives which is a synthesis of current approaches to knowledge acquisition.

We begin with an analysis of the current approaches to knowledge acquisition with respect to strategic knowledge, and then proceed with a description of our methodology and its application to a hypothetical domain. We conclude by relating the sources of power of the method to the properties of the current approaches.

2 Analysis of current approaches to knowledge acquisition

Current approaches to knowledge acquisition can be grouped into three areas: *traditional knowledge engineering*, in which a knowledge engineer builds a program based on interviews with domain experts; *interactive knowledge acquisition tools*, which provide a structured interface for experts and knowledge engineers to build knowledge bases within pre-designed architectures; and *machine learning*, in which learning programs, designed by knowledge engineers, produce operational representations of knowledge from examples of expert performance. In the following sections we examine each from the perspective of acquiring strategic knowledge.

2.1 The knowledge engineering approach

Traditional knowledge engineering methodology puts the burden of incorporating strategic knowledge into an expert system on the knowledge engineer. For problem solving tasks dominated by substantive knowledge, such as the flow of inference in heuristic classification (Clancey, 1985), the strategy is implicit in a domain-independent procedure (the "control strategy") for applying domain knowledge. While this practice makes it easier to acquire substantive knowledge, it prevents the expert from directly contributing strategic expertise.

Recent knowledge engineering research has identified classes of "generic" problem solving tasks and methods, in which a task-specific strategy is incorporated into the method for applying domain knowledge, and the kinds of knowledge appropriate for the task are made clear (Chandrasekaran, 1986; Clancey, 1985; McDermott, 1986). The corresponding *task-level architectures* facilitate knowledge acquisition by providing a higher level of abstraction, more natural to the task, than the implementation constructs used by knowledge engineers (Bylander and Chandrasekaran, 1987; Gruber and Cohen, 1987a).

Most existing task-level architectures provide a control strategy / problem solving method[1] in which the course of action is determined by the structure and content of the substantive knowledge. Again, this relieves the expert of the burden of specifying the control of the system, yet makes it difficult to represent strategic knowledge. For example, an architecture for hierarchical classification, CRSL (Bylander and Mittal, 1986), performs data-gathering actions in an order reflecting the hierarchical structure of the hypothesis space. To represent an alternate strategy of data-gathering based on features of the potential data or data-gathering actions, rather than the hypotheses that the data explores, the knowledge engineer would use implementation-level constructs such as message passing. An exception is the ACCORD framework, a task-level language for specifying the control of search in arrangement-assembly problems (Hayes-Roth, 1987). By providing an explicit representation for search control knowledge, ACCORD makes it easier to write expert systems that incorporate domain-specific strategy.

[1] A "method" for a task is a specialization of weak methods, such as generate and test, that performs task-specific operations, such as "add a part to a configuration" and "test it against engineering constraints." The term is also used to refer to the procedure for applying domain knowledge, which is often the top level control loop of an expert system.

A task-level architecture to facilitate the acquisition of strategic knowledge (as we have defined it) would provide the terms for specifying domain-specific strategies for choosing among actions under multiple goals. The architecture is a necessary step, but is not sufficient to fully address the knowledge acquisition problem. While the architecture provides the terms for strategic knowledge, a knowledge engineer implements the strategy in the system.

2.2 Interactive knowledge acquisition tools

The second line of research on the knowledge acquisition problem is the development of interactive tools that help a knowledge engineer or expert build and maintain a knowledge base. The simplest of such tools are editors that specialize in the syntax of the knowledge representation formalisms (e.g., rule editors) (Buchanan, 1984). More advanced tools take an active role in helping the expert identify the knowledge needed to get a system to exhibit the proper behavior. TEIRESIAS (Davis, 1976), for example, helps the expert debug the knowledge base by identifying missing and incorrect rules. Recently, a family of tools have been developed that are specialized for the kinds of knowledge required for a particular task, using the corresponding task-level architecture (Boose, 1986; Eshelman and McDermott, 1987; Kahn, Nowlan, and McDermott, 1984; Marcus, 1987; Musen, Fagan, Combs, and Shortliffe, 1987). For example, a knowledge acquisition tool for building diagnostic expert systems can request the names of diseases and relevant symptoms from the user, offering some guidance about how to express the knowledge in the appropriate formalism. The key to the success of these interactive tools is that they use knowledge about the task and a particular problem solving method supported by the architecture (McDermott, 1986). This has two primary benefits for knowledge acquisition. First, the interface between the human and machine can be structured to acquire knowledge relevant to the task and only elicit knowledge in the form useful to the method. Second, since it is designed to work with a specific problem solving method, and therefore knows how the method applies domain knowledge, the tool can analyze the user's input for completeness and consistency. In effect, the knowledge acquisition tool reduces the task of building a knowledge base to the task of instantiating known representations for a specific domain.

Interactive knowledge acquisition tools are still limited by the representation in which experts ultimately encode their knowledge. An interactive tool can help the expert express domain knowledge to be used by the method, but it may require knowledge in a form that is inherently difficult to elicit from experts. For example, the MORE (Kahn, et al., 1984) system helps experts fill out an inference network with weighted links associating data and hypotheses. But experts have difficulty providing the numbers. So MORE's successor, MOLE, allows the expert to describe the intended behavior of the network on particular cases, and the tool infers a set of weights. A similar advance is needed to automate the acquisition of strategic knowledge. Representing strategy involves specifying how to choose an action based on subtle tradeoffs among multiple goals that vary in different situations. Although an expert makes strategic decisions every day, it may be difficult to express it explicitly in an operational form such as priorities on an agenda. Thus the challenge is to build a tool that can elicit the strategic knowledge in a natural form for experts and still be able to produce a knowledge base that is operational.

2.3 Machine learning *qua* knowledge acquisition

The third stream of knowledge acquisition research is machine learning. Machine learning promises a way to get computers to behave as we wish without programming them (or, equivalently, without having to build and debug complex knowledge bases). Machine learning can facilitate knowledge acquisition by transforming knowledge that is in a readily available form, such as examples, into a more useful form, such as diagnostic rules.[2] If experts could simply *show* a machine what to do, rather than program it, then the knowledge acquisition problem would be solved.

Machine learning research has produced powerful techniques for inducing generalizations from examples. For the task of simple classification, it is feasible to use a learning algorithm to transform a set of examples (i.e., values of input variables paired with their associated class) into classification rules using the techniques of the concept learning research (Carbonell, Michalski, and Mitchell, 1983). For any application of machine learning to knowledge acquisition, however, somebody has to set up the learning problem for the induction algorithm: designing a representation for examples and generalizations, defining all of the terms in those languages, encoding a set of training examples in the representation, and providing background knowledge (Michalski, 1983) that guides the induction algorithm to choose the right generalizations from the potentially infinite set of possibilities. This can require a significant knowledge engineering effort when the task is more complicated than simple classification.[3]

To summarize, the current approaches to knowledge acquisition each contribute partial solutions to the problem of acquiring strategic knowledge. The knowledge engineering approach suggests an architecture in which a system can be controlled by domain-specific strategic knowledge that is represented explicitly. A task-specific interactive tool can help elicit strategic knowledge of the kind required by the architecture. Yet it seems that the operationality constraint on the representation of strategy — that it be in a form that can run in the architecture — makes it difficult to acquire directly from experts. Learning techniques have complimentary characteristics: while they make it possible for experts to provide their knowledge in an easily available form (examples), they require a significant foundation of knowledge engineering.

The question is whether these approaches can be integrated to address the problem of acquiring strategic knowledge. To explore this hypothesis we propose a knowledge acquisition methodology embodied by an interactive tool that draws from from each approach, automating

[2]Dietterich (Dietterich, 1986) distinguishes between learning new information at the knowledge level (knowledge outside the deductive closure of the learner's inputs) and learning more efficient representations of existing knowledge at the symbol level. Most machine learning techniques support learning at the symbol level, which is useful to knowledge acquisition precisely because it *transforms* the representation. Acquiring new knowledge is difficult for stand alone learning programs but relatively easy for an interactive knowledge acquisition program: it can ask the expert.

[3]The role of the knowledge engineer is even more pronounced in the deductive learning techniques such as explanation-based generalization (Mitchell, Keller, and Kedar-Cabelli, 1986), where the completeness of the "domain model" is critical to the success of the learner.

much of what is currently performed by knowledge engineers, and synthesizing interactive and automatic learning techniques.

In the following section we introduce the approach, and then give a short example illustrating a knowledge acquisition session between and expert and an interactive knowledge acquisition tool. A more detailed account is available in (Gruber, 1987).

3 The ASK methodology

Any approach to knowledge acquisition must provide a bridge between what the expert can express and what the program needs to run. Conventionally, a knowledge engineer bridges the gap between expert and machine. We present a tool, called the knowledge acquisition *assistant*, that automates some of the knowledge engineer's task. The assistant is the cornerstone of a methodology for knowledge acquisition we call ASK, for "Acquisition of Strategic Knowledge." Simply put, the approach requires coming up with a way to elicit strategy from experts, designing an architecture to support strategic knowledge, and developing a technique for mapping the information obtained from experts onto the representations provided by architecture. The three components of the methodology are defined as follows:

1. **The acquirable form:**
   A medium with which experts can describe their strategic knowledge

2. **The operational form:**
   A formalism for representing strategic knowledge that can be executed in a expert system

3. **The transformation from acquirable to operational:**
   A method for transforming knowledge in the acquirable form into the operational form (operationalization) and making it generally useful (generalization)

We describe each component in turn, and then illustrate the methodology with a hypothetical example.

3.1 An acquirable form of strategic knowledge: the justification language

An acquirable form of strategic knowledge is a way for experts to convey their strategic expertise to the knowledge acquisition assistant without having to speak in operational formalisms. In ASK, strategic knowledge is acquired using a *justification language* — a medium for expressing the reasons underlying strategic choices made in individual cases. The justification language is supported by an interactive interface that elicits examples of strategic choices and the justifications for making them in the context of actual cases of problem solving. The expert enters

a case by specifying the sequence of actions taken and their results. The assistant asks the expert to annotate each action with a justification. The justification consists of a list of *control parameters*, factors that were relevant in the decision to choose that action. The control parameters are the terms with which to answer the question "Why did you choose this action?" The assistant presents a set of control parameters to the expert, such as various costs of performing an action, and its potential effect on the current diagnosis. The expert chooses from that set, or may define a new control parameter. In effect, the question "Why did you choose this action?" is structured as "What control parameters on this list were relevant to these actions?". Control parameters are implemented as relations over objects in the substantive knowledge base, such as `cost-of(action)`.[4] While the expert chooses a relation by name ("cost of action"), the assistant finds the objects to which it applies in the given context ("this action"). The instantiation of relations on the individual objects form statements in the justification language that serve as reasons for making a strategic choice ("because this action is free"). For example, if the expert justifies the choice of action Action_1 over action Action_2, then the statement in the justification language is a conjunction of "features" of the actions, where the features are facts such as "cost-of(Action_1) = free" and "diagnostic-for(Action_2,Hypothesis_1)".

ASK provides a mechanism for experts to define new control parameters in the justification language. Some control parameters are difficult to implement and must be built by the knowledge engineer who designs the knowledge system architecture. But many factors relevant to strategic choices are simple predicates over objects in the substantive knowledge base. If an expert needs to make a distinction between two actions while justifying a case, and the available control parameters are inadequate, he or she can define new terms by instantiating classes of control parameters. Although the knowledge engineer who designs the architecture for a task cannot anticipate the actual domain-specific names of things that the expert will use, the engineer can design classes of control parameters from which to define new terms. Suppose, for example, the expert needs to characterize a set of actions by their relative safety in order to decide among them, but the term "safety" is not defined as a control parameter. Conventionally, the expert would have to call in the knowledge engineer. In ASK, however, the safety function can be easily defined by the expert; it is a simple attribute, whose value on the actions of the domain can be supplied by the expert. New terms such as `safety` can be supported by the architecture in the sense that they are instances of a class or type of functions defined over the objects representing actions.

The justification language is acquirable because the assistant provides a constrained user interface. (Figure 1 in the next section shows part of a screen for the example presented in the following section.) Justifications can be formed from existing control parameters by selecting them from menus. The definition of a new control parameter can be acquired in the style of filling in a form, because control parameters are represented as frames in an inheritance hierarchy, with the names and possible values of the slots of the new parameter inherited from its class frame. For example, the expert can define the term `safety` by indicating that it is in the class `attributes-of-actions`. A frame editor can prompt for the

[4] They are described as relations because they may involve other objects in the state of the problem solver, such as hypothesis or goals.

defining characteristics of `safety`: whether it is static (true for all time) or dynamic (calculated from the values of objects in working memory), the data type and range of values it may assume (e.g., the set {`perfectly-safe`, `safe`, `moderately-dangerous`, `dangerous`, `life-threatening`}), and its domain — the subset of actions for which the function makes sense (e.g., the subclass of actions called `trial-therapies`). The information needed by the form-filling interface is inherited from the control parameter class by a frame system, and the form-filling editor is just a frame editor configured to acquire the kinds of slots needed to define a control parameter. The ability to define new control parameters and classes of control parameters is a capability of the MU architecture (Cohen, Greenberg, and Delisio, 1987), upon which ASK is built.

Although the expert uses a language of terms that are defined in a knowledge base, statements in the justification language are not sufficient to specify strategies for programs. The next section describes the representation that can be used to control systems for the task of prospective reasoning (Cohen, Greenberg, and Delisio, 1987).

3.2 The operational representation of strategic knowledge: strategy rules

The operational representation is a formalism for building a knowledge base of strategic knowledge. In ASK the representation is in the form of ***strategy rules***. Strategy rules capture the strategic knowledge underlying the choice of actions in a domain. They explicitly represent local strategic decisions, such as the advice to rule out dangerous diseases first, and of global tradeoffs between multiple goals, such as the judgment to use an invasive diagnostic test when the potential cost of not having the information provided by the test is serious.

Strategy rules associate strategic situations with recommendations about which actions to take. They are similar to the metarules used to represent the diagnostic procedure in NEOMYCIN (Clancey, 1984) and the productions that create preferences in SOAR's decision procedure (Laird, Newell, and Rosenbloom, 1987). An example strategy rule is "if a life-threatening disease is suspected, favor actions that can help rule out that disease over actions that may provide evidence for less dangerous diseases." The left hand sides of rules represent strategic situations, pattern matching expressions over the state of objects in the working memory of the problem solver, such as the current set of active hypotheses, and characteristics of entities relevant to the diagnosis, such as the inherent dangerousness of a disease or the cost of giving a test. These expressions are formed from the same relations that define control parameters. For example, the relation `life-threatening-disease` is used as a left hand side term and as a control parameter. The right hand side represents a recommendation for choosing among actions. Three types of right hand side recommendations correspond to three kinds of strategy rules. *Proposing* rules indicate which actions are plausible candidates in a current strategic situation. *Filtering* rules indicate a situation in which a proposed rule should not be chosen. *Preference* rules specify a preference of one action over another. The left hand side of a preference rule typically compares the values of both actions in a specified context. The effect of the recommendations of strategy rules is to select the set of most preferred, acceptable, plausible actions.

A simple interpreter called the *skeletal control strategy* applies strategy rules to control the execution of an expert system. The skeletal control strategy matches rules to diagnostic situations and merges the recommendations to decide which action to take next. The procedure first runs all matched proposing rules to produce a set of plausible actions, applies any matched filtering rules to remove unacceptable actions, and then orders the remaining set using the preference rules. Preferences are not total orderings, so a lack of a preference between a pair of actions puts them in the same equivalence class with respect to a single preference. Multiple preferences are merged by intersecting the most-preferred equivalence class under each preference. If two strategy rules of any type match, and one is strictly more general than the other, then the more general rule is "shadowed" by the more specific rule (it is considered not-matched). Generality is defined as the subset relation over the control parameters mentioned in the left hand side expressions.

If the final set of actions is empty, perhaps because of a conflicting set of preferences, the skeletal control strategy halts, indicating that there is insufficient strategic knowledge to proceed. Otherwise the set of chosen actions are executed by the system. A typical action involves gathering some data, and after it is executed the effects of the new data are propagated throughout working memory.[5] The skeletal control strategy repeats this loop of matching strategy rules, merging their recommendations, and executing the chosen actions.

3.3 Transforming justified actions into strategy rules

The process of transforming "advice" as it is elicited from the expert into the operational representation is operationalization (Mostow, 1983), and it is performed by the assistant. The operationalization task for ASK is to transform justified actions into strategy rules. The assistant uses knowledge about the method of applying strategy rules (the skeletal control strategy) and the definitions of control parameters to transform justified cases of strategic decisions into highly specific but executable strategy rules. It uses inductive learning techniques to generalize specific examples (and specialize overly general rules) into strategy rules that work in situations other than the training cases. The expert participates directly in the learning loop, providing the vocabulary (new control parameters) for describing examples in the "description language"[6] and by evaluating generalizations proposed by the assistant. The assistant plays an active role in eliciting examples, helping the expert formulate justifications, matching expert choices against the existing strategy rules, and formulating new strategy rules to account for the differences.

The assistant performs credit assignment by analyzing the mismatch between the expert's justified choices and the choices taken by the current strategic knowledge base. The analysis is based on knowledge about the behavior of the skeletal control strategy.[7] If, after applying the

[5] This style of performing inference, in which the effects of new data are automatically propagated along the paths of inferential relations in a symbolic inference net, was used in the MUM expert system for prospective diagnosis in medicine (Cohen, Day, Delisio, Greenberg, Kjeldsen, and Suthers, 1987), and is supported by the MU architecture. In rule-based architectures, the propagation can be achieved with forward chaining.

[6] In the concept learning literature, the representation for learned concepts is called the description language.

[7] This analysis is similar in style to that performed by TEIRESIAS on backward-chaining rules (Davis, 1976).

proposing, filtering, and preference rules, the set of chosen actions is empty, then the assistant assumes there is *missing* or *incorrect* strategic knowledge. Given the expert's justified choice, the assistant considers candidates for blame: (i) a missing proposing rules that would have proposed the expert's action; (ii) an overly general filter rule (when the expert's choice was filtered); or (iii) overly general or conflicting preference rules (when the expert's choice was dominated by a preference for another action). If, instead, there are multiple actions chosen by the system, the assistant considers these cases: (i) the actions are all equally preferable, so no new knowledge is needed; (ii) there is a missing filtering rule that would have filtered some of the actions; or (iii) there is a missing preference rule that would have caused some of the actions to be removed from the choice set.

From the credit assignment analysis and the justified choices, the assistant can propose the right hand side of a strategy rule (e.g., "To account for this case, propose a rule that prefers Action_{13} over Action_{20}."). The next step is to formulate the left hand side of the rule by operationalizing justifications into pattern matching expressions. The assistant can perform this transformation because the justification language and pattern matching expression in strategy rules share terms — the control parameters. The same relations that provided the expert with a language of justifications for choosing an action are also used in the left hand sides of strategy rules. They are, after all, operational primitives; the system can run them as tests on the state of the diagnosis. If the expert stated that factors F_1 through F_n were relevant to the choice of action A, then the strategy rules that recommend A would contain some expression over the relations defining the factors F_i. For example, if the expert said that F_1 (the cost of a test) and F_2 (the false positive rate of that test) and F_3 (the dangerousness of the current hypothesis) are the factors influencing the decision to take action A, then the left hand side of a corresponding strategy rule is the conjunction of the values of those factors in the given case: $F_1(A) = x \wedge F_2(A) = y \wedge F_3(H) = z$, where x, y, and z are the exact cost of the test A, the false positive rate of A, and the degree of criticality of the disease hypothesis H. The action part of the strategy rule is a recommendation to choose A.

Newly formed strategy rules are operational yet not general. The assistant performs generalization by analyzing the new, very specific strategy rules and the existing body of strategy rules to find plausible generalizations. Possible generalizations are simple to generate; well-known syntactic operations from machine learning can transform an expression like the left hand side of a strategy rule into an expression that holds for a larger set of situations (Michalski, 1983). For instance, the ='s in the previous expression could be generalized to $>$ or $\in$. One or more of the conjunctive terms could be dropped (e.g., leaving off the $F_3 = z$). Or the constant A could be turned into a variable, generalizing the expression to apply to *any* action for which the applicable conditions F_1 and F_2 hold. However, the difficulty with syntactic generalization is determining which generalizations to make. On deductive grounds the generalizer could eliminate any generalizations that would conflict with rules derived from the cases. Some constraint on generalization can come from the knowledge about the terms; if F_1 is an integer function, then a reasonable generalization is a test for a range of values, whereas Boolean predicates can only be dropped or added. But the major source of constraint on generalization comes from the expert who determines the set of relevant factors in each situation and confirms or rejects

generalizations proposed by the knowledge acquisition assistant.

This process of eliciting cases, supplying justifications for each choice, transforming them into strategy rules, and finding more general rules is orchestrated by the assistant. It prompts the expert for information and poses questions about what the expert has said. For example, if the expert selects factors F_1 through F_3 to justify action A, yet those factors are insufficient to distinguish A from another action A', then the assistant would ask the expert to select other factors which will distinguish A from A'. The dialogue between the expert and the knowledge acquisition assistant is continued until the strategic knowledge base can account for the choices and justifications of the expert demonstrated in the cases.

The ASK approach assumes that a knowledge engineer first designs an architecture for supporting strategy, including the implementation of some control parameters general to the task.[8] It also assumes that the substantive knowledge of the domain, including the set of possible hypotheses, actions, data and intermediate conclusions, the associations or inferential relationships between data and conclusions, and the possible effects of actions (e.g., the data they produce and costs they incur) have been already acquired. Furthermore, it assumes that the knowledge engineer and/or expert have provided some initial strategy rules that are very general. For example, one rule that is predefined for the task of prospective diagnosis is "propose actions that provide evidence for or against the triggered hypotheses", where the triggering relationships are acquired from the expert.

4 A simple example

Consider a rather simple but common diagnostic problem: a car that will not start. Imagine an expert system with all the substantive knowledge of car repair, the sort of information found in shop manuals. The system knows about all the possible actions that a mechanic might do to a car (and their prices). It knows the data that actions produce, and the diagnostic significance of the data. The system does not know how to diagnose a car problem; it lacks strategic knowledge. Imagine a mechanic sitting down with the knowledge acquisition assistant to teach the system how to to do it. And try not to laugh.

In the first step the assistant asks the expert to enter a case – a *work-up* for a particular car. A work-up is a series of actions taken by a diagnostician in a particular case. The work-up for diagnosing the problem with a car that will not start is shown in Figure 1. The expert enters the actions as nodes, and labels the arcs with the outcome of previous actions. (The assistant provides an interface by which the expert can select from the set of actions, hypotheses, data, etc. with a mouse.)

In the next step, the expert and assistant produce a justification for each action. The first strategic action is to check the headlights. From the substantive knowledge, the assistant

[8] Our implementation is supported by the MU software environment that makes it easy to define functions and relations to use as control parameters.

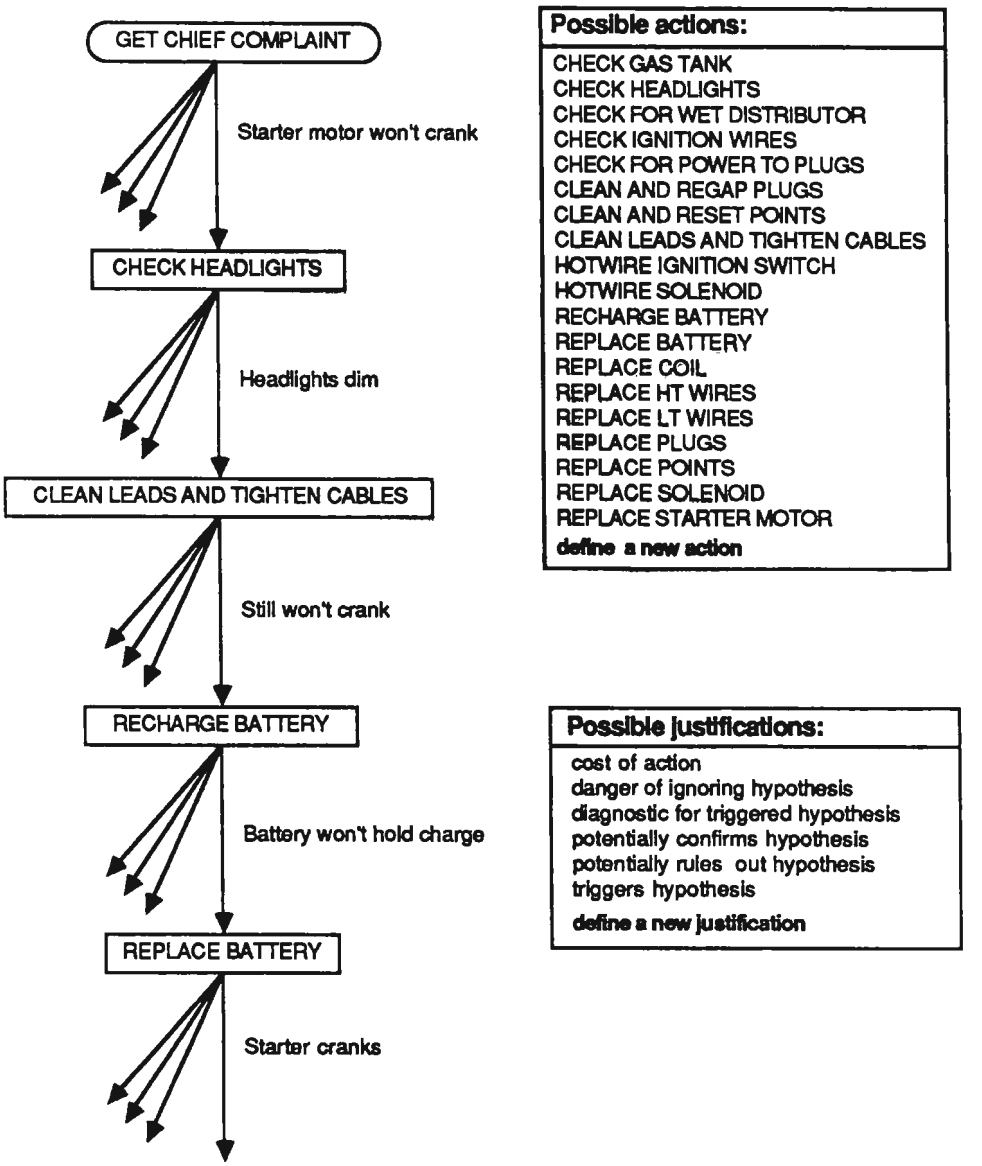


Figure 1: A work-up for diagnosing a car that won't start

This is a case of prospective diagnosis used to acquire strategic knowledge. Each node is an action chosen by the expert. Each labeled arc is the actual outcome, and the extra arrows represent possible alternative outcomes. The set of possible actions and current state of the diagnosis are available from menus. The expert annotates each action with reasons for choosing it, selected from the list of possible justifications.

knows that if the headlights were bright, the result of checking the headlights would rule out several hypotheses related to inadequate power supply. The assistant provides a set of possible justifications, based the control parameters and strategy rules that are currently defined for the domain. One of these, `diagnostic for triggered hypotheses`, is a relation that holds between actions and the hypothesis for which they provide diagnostic evidence. This control parameter is represented in the left hand side of a strategy rule as a conjunction of two relations:

```
    if (triggered $hypothesis)
       (diagnostic-for $action $hypothesis)

  then (propose $action)
```

It says "try actions with possible outcomes that would provide evidence for and against hypotheses that are under consideration."

The assistant finds that the `diagnostic-for` justification is also true of other actions, in particular, recharging the battery and replacing it both have the potential effect of ruling out triggered hypotheses. So the assistant asks the expert for additional justifications for the `check-headlights` action that will distinguish it from `recharge-battery` and `replace-battery`. The assistant provides a set of possible answers by generating the values of all control parameters at the point where the `check-headlights` action was chosen. The expert picks `cost-of-test` from among them. The justifications for the check-headlight action now looks like this:

```
(triggered dead-battery)
(potentially-rules-out check-headlights dead-battery)
(cost-of-test check-headlight 0)
```

The assistant now compares the `cost-of-test` attribute of the three actions and finds an expression that distinguishes `check-headlights`: its cost is $0, where the cost of `recharge-battery` and `replace-battery` are $5 and $50 respectively. The assistant now operationalizes a rule that says, "if two actions can both rule-out hypotheses and one costs $0, choose that one." This preference is represented by the strategy rule:

```
    if (potentially-rules-out $action1 $hypothesis1)
       (potentially-rules-out $action2 $hypothesis2)
       (cost-of-test $action1 0)
       (cost-of-test $action2 (not-eql 0))

  then (prefer $action1 $action2)
```

The assistant and expert negotiate in a similar manner over the reasons for choosing to recharge the battery before buying a new one. The assistant proposes the obvious (syntactic) generalization that says to prefer cheaper actions. But the expert chose to not replace the solenoid

before the battery, which costs more. The assistant found this out by proposing hypothetical situations, presented graphically by extending the work-up tree along the dangling left branches leaving each node, and asking the expert to justify *not* choosing the alternative actions. In the hypothetical case, both `replace-battery` and the alternate `replace-solenoid` are diagnostic for triggered hypotheses and potentially confirm them (and solve the problem).[9] So the assistant asks for a justification that will distinguish the solenoid action. Answering the assistant by selecting from lists of relations, the expert indicates that `bad-solenoid`, the hypothesis potentially confirmed by the `replace-solenoid` action, is currently only triggered, whereas the hypothesis `battery-needs-replacing` is strongly supported by the "battery won't hold charge" finding. With the expert's feedback, the assistant can form a rule that says that the preference for cheaper actions is less important than the preference for actions that confirm hypotheses that are strongly supported.

5 Discussion

ASK draws from and contributes to each of the areas of work in knowledge acquisition. As a synthesis of techniques, the research is an attempt to demonstrate the sources of power in various approaches, as well as to establish a methodology for combining them to solve a knowledge acquisition problem.

First, it is built upon the knowledge engineering support provided by a task-level architecture: display utilities, editors, runtime support (e.g., inference networks), all configured to the representational needs of the task (Gruber and Cohen, 1987b). ASK is designed to clarify the role of the task-level knowledge in the architecture on the knowledge acquisition method. The sources of task knowledge used in the ASK approach can be stated explicitly:

- Definitions of the major classes of objects in the knowledge base, such as hypotheses, actions, data, and intermediate conclusions. These classes serve as the domains of relations that implement control parameters.

- Definitions of relations among objects that are intrinsic to the task. For prospective diagnosis these include evidential support, triggering, and the diagnosticity relations such as `rules-out-triggered-hypothesis`.

- Definitions of the classes of control parameters assumed to be relevant to strategy in a domain, such as `attributes-of-actions`.

- A method for applying strategy rules — the skeletal control strategy for matching rules to diagnostic situations and merging their recommendations to decide which action to take next.

[9] In this domain, "confirm" often means "fixes the problem," since most of the hypotheses are of the type "*device* needs replacing."

- A small number of general strategy rules to bootstrap the process.

A consequence of making the built-in task-level knowledge explicit is that it defines the roles of the knowledge and expert along new boundaries. The expert is responsible for designing a task architecture and implementing the basic task-specific control parameters; the expert is responsible for providing instructive cases, annotating them with justifications, and extending the vocabulary of control parameters as needed, within the space laid out by the engineer (i.e., the space is defined by the classes of control parameters).

ASK supports the conclusion that there is power in specializing a knowledge acquisition tool to a particular task and method, as demonstrated by ETS, MOLE, SALT, and OPAL (Boose, 1986; Eshelman and McDermott, 1987; Marcus, 1987; Musen, et al., 1987). Perhaps the ability to represent strategy as domain knowledge itself may allow knowledge engineers to explore a space of strategies within classes of tasks. Since many problem solving methods are in a sense strategies (e.g., for search), the approach may be used to devise new task-specific methods.

Finally, ASK integrates machine learning techniques with the traditional paradigms of knowledge acquisition. The interactive dialogue between expert and knowledge acquisition assistant can be viewed as an inductive learning algorithm, where the human is acknowledged as an integral player. The expert supplies the examples: sequences of chosen actions. The assistant poses hypothetical examples to test the boundaries of its strategy rules. (This is called "perturbation" (Kibler and Porter, 1983).) The assistant performs credit assignment with a case analysis based on the skeletal control strategy. The assistant generalizes using well known techniques for inductive generalization, such as dropping conditions and turning constants into variables. The bias (Mitchell, 1980) or background knowledge (Michalski, 1983) that directs generalization is provided by the knowledge engineer in the architecture. The expert serves as critic (Dietterich and Buchanan, 1981), and the evolving knowledge system acts as a performance element.

The ASK methodology addresses three important problems for machine learning that arise in the practice of knowledge acquisition:

- **Encoding examples.** In most machine learning programs it is assumed that the training data is available to the learner in the same language that is used in generalization; this is called the "single representation trick" (Dietterich, 1982). To engineer out the problem of multiple representations would beg the question for knowledge acquisition, since it is assumed that representational mismatch *is*, in large part, the obstacle to overcome. ASK deals with this problem head-on by providing a medium for experts to explain their actions (the justification language) without requiring it to have the same constraints as an operational language (strategy rules).

- **Operationalization.** ASK treats the problem of making expert advice operational as the transformation of knowledge in the justification language into the language of strategy rules. This is akin to mapping domain-specific descriptions of goals onto a representation

of weak methods, as in FOO (Mostow, 1983). Our solution may be called the "intersecting representations trick" — the acquirable and operational representations of strategic knowledge share terms (the control parameters).

- **New Terms.** The new term problem occurs in a learning system when a desired generalization cannot be represented in the description language. Utgoff (Utgoff, 1986) demonstrated that a solution should include a means for recognizing when a new term is needed, a method for generating the new term, and a way to operationalize that term (e.g., make it work like other terms in the description language). The justification dialogue in ASK makes it clear when a new term is needed: when there are no control parameters defined that can distinguish two actions which the expert says should be distinguished. The expert can define new terms by instantiating predefined classes. They are operational because the architecture defines the implementation code for classes of control parameters which is inherited by new control parameters.

Acknowledgments

Paul Cohen collaborated in developing the underlying ideas for this work, and reviewed the draft. JeFF Delisio and Mike Greenberg implemented the MU architecture, providing a productive environment for experimentation.

References

Boose, J. H. (1986) *Expertise Transfer for Expert System Design*, New York: Elsevier.

Buchanan, B. G., Barstow, D. K., Bechtel, R., Bennett, J., Clancey, W., Kulikowski, C., Mitchell, T., & Waterman, D. A. (1983) Constructing an Expert System. In F. Hayes-Roth, D. A. Waterman, & D. B. Lenat (Eds.), *Building Expert Systems*, Reading, MA: Addison-Wesley.

Buchanan, B. G. & Shortliffe, E. H. (Eds.) (1984) *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Reading, MA: Addison-Wesley.

Bylander, T. & Mittal, S. (1986) CRSL: A language for classificatory problem solving and uncertainty handling. *AI Magazine*, **7**(3), 66–77.

Bylander, T. & Chandrasekaran, B. (1987) Generic tasks in knowledge-based reasoning: the 'right' level of abstraction for knowledge acquisition. *International Journal of Man-Machine Studies*, **26**(2), 231–244.

Carbonell, J. G., Michalski, R. S., & Mitchell, T. M. (1983) An overview of machine learning. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach.* Palo Alto: Tioga, 3–24.

Chandrasakeran, B. (1986) Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. *IEEE Expert*, Fall, 23–30.

Clancey, W. J. (1984) Acquiring, representing, and evaluating a competence model of diagnosis. KSL Memo 84-2, Department of Computer Science, Stanford University. To appear in Chi, Glaser, and Farr (Eds.), *Contributions to the Nature of Expertise*, in preparation.

Clancey, W. J. (1985) Heuristic Classification. *Artificial Intelligence,* **27**(3), 289–350.

Cohen, P. R., Day, D., Delisio, J., Greenberg, M., Kjeldsen, R., & Suthers, D. (1987) Managing uncertainty in medicine. *International Journal of Approximate Reasoning*, **1**(1).

Cohen, P. R., Greenberg, M., & Delisio, J. (1987) MU: A development environment for prospective reasoning systems. *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, Washington, July, 783–788.

Davis, R. (1976) Applications of meta-level knowledge to the construction, maintenance, and use of large knowledge bases. Doctoral dissertation, Computer Science Department, Stanford University. Reprinted in R. Davis & D. B. Lenat (Eds.), *Knowledge-Based Systems in Artificial Intelligence*, New York: McGraw-Hill, 1982.

Dietterich, T. G. (1982) Learning and Inductive Inference. Chapter XIV in *The Handbook of Artificial Intelligence*, Vol. III., P. R. Cohen & E. A. Feigenbaum (Eds.), Reading, Massachusetts: Addison-Wesley.

Dietterich, T. G. (1986) Learning at the knowledge level, *Machine Learning*, **1**(3), 287–316.

Dietterich, T. G. & Buchanan, B. G. (1981) The role of the critic in learning systems. Report STAN-CS-81-891, Department of Computer Science, Stanford University, December.

Dietterich, T. G. & Michalski, R. S. (1983) A comparative review of selected methods for learning from examples. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach.* Palo Alto: Tioga, 41–81.

Eshelman, L. & McDermott, J. (1987) MOLE: A tenacious knowledge acquisition tool. *International Journal of Man-machine Studies*, **26**(1), 41–54.

Gruber, T. R. & Cohen, P. R. (1987a) Design for acquisition: principles of knowledge system design to facilitate knowledge acquisition, *International Journal of Man-machine Studies*, **26**(2), 143–159.

Gruber, T. R. & Cohen, P. R. (1987b) Knowledge engineering tools at the architecture level. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, August.

Gruber, T. R. (1987) A methodology for acquiring strategic knowledge from experts. EKSL Report 87-1, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts.

Hayes-Roth, B., Garvey, A., Johnson, M. V., & Hewett, M. (1987) A modular and layered environment for reasoning about action. Knowledge Systems Laboratory Report No. 86–38, Stanford University, April.

Kahn, G., Nowlan, S. & McDermott, J. (1984) A foundation for knowledge acquisition. *Proceedings of the IEEE Workshop on Principles of Knowledge-base Systems*, Denver, Colorado, December, 89–98.

Kibler, D. & Porter, B. (1983) Perturbation: A means for guiding generalization. *Proceedings of Eighth International Joint Conference on Artificial Intelligence*, 415–418.

Laird, J. E., Newell, A., & Rosenbloom, P.S. (1987) Soar: An architecture for general intelligence. *Artificial Intelligence*, **33**(1), 1–64.

Marcus, S. (1987) Taking backtracking with a grain of SALT. *International Journal of Man-machine Studies*, in press.

McDermott, J. (1986) Making expert systems explicit. *Proceedings of the Tenth Congress of the International Federation of Information Processing Societies*, Dublin, Ireland.

Michalski, R. S. (1983) A theory and methodology of inductive learning. *Artificial Intelligence*, **20**, 111-161.

Mitchell, T. M. (1980) The need for biases in learning generalizations, Technical Report CBM-TR-117, Rutgers University.

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986) Explanation-based generalization: a unifying view. *Machine Learning*, **1**(1), 47–80.

Mostow, D. J. (1983) Machine transformation of advice into a heuristic search procedure. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach.* Palo Alto: Tioga, 243–306.

Musen, M. A., Fagen, L. M., Combs, D. M., & Shortliffe, E. H. (1987) Using a domain model to drive an interactive editing tool. *International Journal of Man-Machine Studies*, **26**(1), 105.

Utgoff, P. (1986) *Machine Learning of Inductive Bias.* Norwell, Massachusetts: Kluwer Academic.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

ASTEK: A Multi-Paradigm Knowledge Acquisition Tool for Complex Structured Knowledge

Chris Jacobson

Michael J. Freiling

Tektronix Laboratories
Tektronix, Inc.
P.O. Box 500 MS 50-662
Beaverton, OR 97077
chrisj@tekchips.tek.com
freiling@tekchips.tek.com

ABSTRACT

Knowledge based systems can require large, highly complex and varied forms of knowledge. An effective knowledge acquisition tool to support such a system should allow the user to transfer and manipulate the different forms knowledge in a manner that is clear and intuitive. ASTEK is a knowledge acquisition tool that provides multiple paradigms for knowledge editing while maintaining a single, consistent framework designed using natural language discourse concepts.

INTRODUCTION

Knowledge acquisition is a process of gathering knowledge from one or more domain experts for utilization in a knowledge based system. To reach the necessary levels of performance , knowledge based systems can require large, highly complex and varied forms of knowledge.

A fundamental tenet of our approach to knowledge engineering is that effective knowledge based systems can and should be built by designing cognitively appropriate knowledge structures, and implementing inference strategies as an operational semantics for these structures.[Freiling1985a, Alexander1986a] There is a tradeoff, however, between the ease with which structures can be expressed by domain experts and the most tractable form for manipulation by an inference engine.[Gruber1987a, Musena] Effective knowledge acquisition tools must support paradigms for knowledge entry that are easy for the domain expert to use and minimize any cognitive burdens. They must then translate the actions within these entry paradigms into structures appropriate for operational interpretation. It is the need for multiple paradigms of knowledge entry that distinguishes our problem from that addressed in similar efforts.[Bobrow1986a]

In this paper we describe a multi-paradigm knowledge acquisition tool called ASTEK (Acquisition of STructured Expert Knowledge). ASTEK is implemented in Smalltalk and runs on Tektronix 4400-series workstations. ASTEK permits

domain-expert users to define structured knowledge fragments in **winus**. Winus combine a window for editing parts of a knowledge structure with a menu for selecting which part to edit. ASTEK attempts to exploit the wealth of current work in natural language systems while allowing the user to define each piece of knowledge using the most natural method of acquisition.

THE PROBLEM

We would like to build tools that domain experts find easy to use in entering the large amounts of knowledge needed for an effective knowledge based system. While domain experts are knowledgeable in their own field, however, they are often novices when it comes to entering large amounts of structured information into a computer system. To support the expert in this endeavor, we feel that an acquisition tool must serve to:

- provide an external form for the knowledge that has clear, intuitive semantics to an expert in the domain
- mediate the knowledge transfer from the external form, that which is natural to the domain expert, to the internal form, that which is most easily manipulated by the inference engine
- support input of knowledge that is "correct by construction"; the acquired knowledge should be constrained to be correct at the earliest possible time
- support knowledge management, such as revision control, and incremental testing; the tool should act as a comprehensive knowledge maintenance mechanism

Our research on knowledge acquisition has focused on acquiring knowledge for building troubleshooting assistant programs.[Freiling1984a] INKA [Phillips1985a, Phillips1985b] was our first attempt to build a useable knowledge acquisition tool. The external form for troubleshooting knowledge was a set of English sentences generated by a context-free grammar.[Freiling1984b] The internal form was a collection of Prolog structures for use by a troubleshooting inference engine. The inference engine processed troubleshooting rules that had an extremely simple structure.

IF <measurement results>
{ WHEN <setup procedure> }
THEN <diagnostic belief>

Because of their simple structure, rules were easily read and written as English sentences. The semantics for troubleshooting rules in English were intuitively clear. Defining the rule language in a context-free grammar insured that rules generated by INKA were correct by construction, at least to the syntactic level. INKA was powerful enough to build troubleshooting knowledge bases for small circuit board fragments.

Subsequent experimentation with INKA and its underlying inference engine revealed several deficiencies when applied to a wider class of troubleshooting problems. First, expert troubleshooting technicians tend to have a high degree of explicitly procedural knowledge about how to perform setup and measurement tasks. This type of knowledge was quite cumbersome to encode in INKA rules, and required a better medium of articulation.

Second, technicians became frustrated with trying to generate a single sentence containing all the information about how to reach one particular conclusion. A technician might generate part of the sentence, and then decide that, rather than being the beginning of a rule description, it was really the middle of a troubleshooting rule (the fragmentation problem below). It was also not possible in INKA to relate parts of one rule (like its conclusion) to any other rule. What the technician wanted to generate was more like a paragraph. The increased structure induced by paragraph-sized descriptions introduced a need for visible cues as to the scope of each paragraph (the utterance planning problem below).

Based on this and other experiences, a more general set of troubleshooting knowledge structures was designed, and a new inference engine, HIPE[Rehfuss1985a, Freiling1986a] was built to process them. HIPE's knowledge structures included explicit procedures for **tests** and **measurements, strategies** to control test selection, and **interpretations** to link measurement results with diagnostic beliefs. With these larger and more complex knowledge structures came the need to rethink our assumptions about the tools necessary for acquiring them.

PROBLEMS OF NATURAL LANGUAGE KNOWLEDGE ACQUISITION

Several problem areas have been identified during our earlier efforts, some of which have been recognized in earlier papers as well. [Bobrow1986a, Normana] These problems have contributed to the development of a set of principles used in the design and construction of ASTEK. Each of the problems and principles will be introduced and briefly discussed.

Multiple External Forms

The principle cause of our shift away from a pure natural language solution has been the need to describe procedural knowledge in a convenient manner. The tree-like structure of diagnostic procedures lies in stark contrast to the inherently linear nature of natural language. Yet in considering the concept of multiple forms we recognized that other forms as well may serve useful purposes. As well as methods to support the expression of natural language sentences and tree structures, such forms might include a menu selection method of acquiring a simple piece of information from a small, fixed set of choices or the fill-in-the-blank method utilized by Musen, et al in OPAL as well as many others.

Natural Articulation Principle

Different syntactic aggregates can best be expressed by using different media of articulation. Like Musen, et al's work in acquiring knowledge for ONCOCIN, we are using a domain model to "solicit and display knowledge" in a manner that is "intuitively understandable" to the domain expert. Our experience, and that of others, has shown that complex knowledge structures decompose into syntactic aggregates, structured groupings of symbols or other knowledge components. Each syntactic aggregate may have a preferred medium of articulation. A date, for example, is composed of a month, a day and a year and is readily formulated for expression using natural language. A troubleshooting procedure, on the other hand, is a tree structure whose

nodes are tests and measurements and whose branches are conditions based on the results of prior tests and measurements. A procedure is cumbersome to describe using a linear tool such as natural language but can be readily expressed and modified with the use of the a visual syntax.

Utterance Planning

With INKA the user is initially faced with a blank screen. INKA tried to be helpful by prompting with all words that could be used to start a sentence, but the users found this unsatisfactory. Once a user has managed to get started with the task of transferring knowledge, a challenge of formulating the knowledge still exists. Humans engaged in conversation do not need to consciously plan a sentence because there are mechanisms in spoken language for extending and redirecting an initial utterance to capture a nearly arbitrary amount of subsequent detail. When the knowledge sought is more structured, however, the expert needs to have an overall feel for all of the components a particular formulation in order to plan the expression appropriately. INKA provides no way to get an overall feeling for the type of expression to be expected.

The Recognition Principle

The medium of articulation should be recognition-based whenever possible. Research into the structure of human computer dialogues has shown that humans avoid the use of English constrictions (such as pronominal references) that would be difficult for a machine to interpret. Allowing the human user to recognize from a checklist the structure elements that need to be filled in makes it possible for the user to break-up the problem of expressing a large structure into simpler problems of expressing each component. Further the user does not have to worry about which connective to use between structural components.

Menus serve a similar function at a lower level. Rather than fish for the lexical item a computer would most readily understand, the user is presented with lexical choices. The user has more of a feeling of being on the right track when one of the choices is appropriate. When none of the choices are appropriate, the user gets a clear signal that things are not going well. In addition menu selection of lexical items saves keystrokes and can lead to reduced time even for the mechanical process of entering an expression.

Fragmentation

INKA expected the knowledge to come in the form of a stream of self-contained sentences, each of which was well-formed. In practice, human users don't seem to seem able to pack all that they want to say into a single sentence. Instead they tend to skip from structure to structure, or to pause to construct a reference to an object that they need to mention in the structure they are building.

The Fragment Expression Principle

When a relevant thought occurs to the user, it should be convenient to express it. The acquisition tool should be able to accept fragments of knowledge from the user without demanding the cognitive burden of formulation needed to construct a complete knowledge structure and without the need to remember the knowledge fragments until such time as the tool desires them. When possible the user should even be able to create a fragment without any stipulation as to its contents, e.g. in forward referencing of a yet to be created knowledge structure.

Reference

It has been noted that for humans,

> "The problem of reference, then, is the problem of specifying for the other person just which concept is meant, either with sufficient precision that the concept can be found, or with sufficient specification that it can be created".[Normana]

The references needed in describing troubleshooting knowledge to a knowledge acquisition tool are definite descriptions involving noun phrases often composed of nominal compounds and with attached prepositional phrases. Spoken descriptions of troubleshooting knowledge also contain many pronominalizations but as has been recently noted, users often prefer to use the full object reference over the more efficient pronoun when communicating with a computer.[Guindon1987a] Objects that need to be referred to in building knowledge structures include both real objects from the task domain and other knowledge structure entities that have been or will be built.

Reference Context

In a reasonably sophisticated knowledge base for a troubleshooter, the number of objects that might be referenced grows to be too large to deal with as a whole. When presenting reference choices to the user, it is necessary to constrain the set of objects to which the user may refer to those in which a reference would be meaningful.

The Reference Principle

The referencing of objects should be expedited. Research on the structure of natural language discourse has demonstrated that the acts of reference to objects and concepts constitutes a major part of discourse. Norman and Rumelhart go as far as to say, "In our analysis of language we assume that every noun phrase - in fact, every sentence - is an expression of reference". An act as pervasive as reference in a discourse tool must be strongly supported.

Open-Endedness

INKA could not make recommendations as to what types of sentences were appropriate to generate given the sentences already constructed. This leaves the human user with an uncomfortable feeling of open-endedness. There is no confirmation that the sentences being generated will add up to anything. An

effective knowledge acquisition tool needs to be able to make suggestions about structures to work on or errors to correct.[Kittoa] To suggest an activity for the user, however, the acquisition tool needs a notion of both the completeness and the correctness of structures to a level beyond that of syntax.

The Correctness Principle

The knowledge entered by the domain expert should be constrained to be correct, to a level beyond that of mere syntax, at the earliest possible time. The user should not have to enter a complex piece of knowledge only to find out much later that an object was referenced that was not valid in the context of the knowledge.

The Completeness Principle

A knowledge acquisition system should be able to identify missing components of knowledge given the current state of the knowledge base. A notion of completeness is necessary largely because the user is allowed to express knowledge in fragments.

A LOOK AT ASTEK

We have discussed some of our past efforts at knowledge acquisition in this paper. As well we have identified some of the weaknesses of our past approach and postulated some principles for a better approach to knowledge acquisition. We would like to now turn to look at our most recent tool for knowledge acquisition, ASTEK. We will discuss winu structures, the media that have been implemented for editing knowledge components and the support in ASTEK for referencing objects and formulating knowledge. The figures referenced are parts of an annotated example contained in the appendix at the end of the paper.

The Framework

ASTEK is implemented in Smalltalk-80 and is a window-based tool. An explicit **focus space** of objects that can be readily referenced is helpful in understanding human utterances in the context of a connected discourse.[Grosz1986a] ASTEK, therefore, supports the user's need to continually refer to objects by maintaining an *explicit* and *visible* object reference stack in a window, and exploiting the concept of "pointing as reference"[Freiling1984a] to allow the user to readily refer to objects present on the screen without having to generate cumbersome referring expressions like "the node whose voltage was measured in the last test that I wrote". Figure 1(a) shows a sample focus space as the technician begins to enter knowledge about the channel reset test. ASTEK also supports descriptive reference to focus space objects in its natural language editor.

When the technician makes a request to the discourse manager to add a new knowledge structure [Figure 1(b)], the appropriate winu is created. A winu combines a window for editing parts of the knowledge structure with a menu for selecting which parts to edit. Selecting a knowledge component (a syntactic aggregate) in the menu pane of the window installs the appropriate editing mechanism (the medium of articulation) in the editing pane. In figure 1(c), the technician expresses the purpose of the test being created. Since the purpose is a free-form comment, the window editor for this slot is just the standard Smalltalk

paragraph editor. All knowledge transfer is performed through manipulation of winus, so ASTEK's overall framework is easy to learn. The ability to design classes of knowledge structures that consist of natural syntactic aggregates and that are assigned the user's choice of a medium of articulation results in a tool that is straight forward to apply to a new domain and that is easy to modify.

The use of winus as the means for the user to define a knowledge structure assures that the user is always aware of the overall knowledge structure to which he is contributing. Each time that an editor is invoked, a specific type of knowledge is known to be expected and its role in the overall knowledge structure is clear. The editing mechanism invoked will be one that is appropriate and lends support to the expression of the information.

The only limit on the number of winus that may be visible on the user's screen at any one time is the size of the screen. The user can readily maneuver from winu to winu as pieces of knowledge occur to him. The only complexity that this introduces is the shift of focus as the user moves to a new structure. ASTEK currently maintains only a single, global object stack for use in referencing but we have seen a need for a more comprehensive scheme of *attentional states* through the introduction of *local focus spaces* as described in Grosz and Sidner's work. Figure 2(a) shows how the discourse manager focus space has changed as a result of beginning work on the channel reset test.

Through the use of graphics, menus and word prediction, many simple pieces of knowledge are expressible without the cognitive burden of sentence formulation. This results in a natural and efficient means of communication between the user and the acquisition tool.

It is interesting to note that the implementation of ASTEK in an object oriented language has been a very natural approach. Object classes exist for each type of knowledge structure and the class "knows" which medium to apply to its components and how to convert the components to the internal form for use by the inference engine. The acquisition environment is a structured interface environment that is independent of any particular set of knowledge classes or media. New classes of structured knowledge can be easily added for nearly any domain without modifying the ASTEK framework.

The Editors

The only limitation to the editing mechanisms that may be utilized by ASTEK are those limitations imposed by the machine on which it is running. The knowledge engineer, with the domain expert, can decide which method of articulation is most appropriate for each component of a knowledge structure. Several media are being used by ASTEK and some of these will be briefly discussed.

The Natural Language Editor - The natural language editor was constructed with the use of a grammar development environment developed in the Tektronix Laboratories.[Messick1986a] KIERYA provides a rich functionality for developing and testing a unification-based grammar.[Kay1985a] The editor offers such features as word prediction, which reduces the effort of sentence formulation, and word completion, for minimization of keystrokes. Figure 2(a) shows the natural language editor prompting the technician for the month as the technician enters the date. Figure 2(b) illustrates the feature structure created as a result of

parsing the entire date entry.

The editor provided with KIERYA also maintains an always-present cursor that turns thumbs-down when the sentence cannot be "understood" and turns thumbs-up when a valid sentence has been completed. This allows the user to detect any input problems as soon as they are created [figure 5(c)].

As mentioned earlier, KIERYA's editor supports descriptive object reference. Partial descriptions of objects such as "the thing connected to the user terminal" are unified with full descriptions of the objects in the local context to determine the correct reference. In the cases where the unification is unable to completely disambiguate the reference, the user is asked to choose from the set of recognizable object descriptions that unified with the partial description. Also in support of the correctness principle, KIERYA's editor often has the objects from the context of the knowledge being edited pre-loaded into the lexicon when the editor has started. Only references to objects that are appropriate in the current context are, therefore, allowed by the parser, and unification disambiguation is the only process that needs to occur after parsing. The disambiguation occurs only over a highly constrained set of objects.

The Tree Editor - The procedures used by the inference engine, HIPE, are hierarchically related references to knowledge structures. Describing a hierarchical structure using a a linear syntax, such as that of natural language, is cumbersome. The tree editor assists this task by utilizing graphical manipulation techniques to convey the structure. Initially, the tree editor is blank, as in figure 3(a). The expert technician chooses to enter a node by selecting from a menu [figure 3(a)], and then stipulates the type of node to add by selecting from another menu [figure 3(b)]. For the procedure tree in our example, the only valid node types are the HIPE structures **strategy**, **test**, **measurement**, and **action**, but the tree editor can be customized for different node types. Figure 7, for instance, shows the interpretation tree, where the nodes are diagnostic beliefs. Each node in the procedure tree represents either a reference to a step in the procedure or to a condition that, based on the results of previous steps, will cause a branch to a specific sub-procedure. In figure 3(c) the technician has already stipulated that he wants to add a node of type measurement, so ASTEK presents a menu of measurements from the focus space. The technician then selects the relevant measurement (in this case **busy_channel**) with the mouse. A summary token for the selected reference is entered into the tree [figure 3(d)]. Note that this new reference to the busy_channel measurement moves it to the top of the discourse manager focus space. Thus, the technician can create a reference to any appropriate structure in the knowledge base with only a few mouse clicks. The top menu item is selected in case the technician wants to create a "forward reference" to a yet unnamed measurement structure.

Another task of the tree editor is to bind local variables to the parameters of procedure steps being referenced. This allows a test that, for example, knows about both a local multiplexer and a remote multiplexer to perform a general baud rate measurement only on the device of interest. In figure 4(a) we see that the technican has established three variables, MUX, NODE, and CHAN, for the test as a whole. When a procedure tree node for the busy_channel measurement is created, the input paramenters for that measurement (MUX, NODE, and CHANNEL) are displayed in a new winu. The tree editor knows that variables are

special entities, and collects all variables referred to in the test structure into a menu, so that input parameters of the measurement instance may be bound with only a mouse click. In the example of figure 4, the only menu variables are the parameters of the **channel_reset** test itself, but in other cases, local variables whose values are assigned in the procedure tree are also included. When the selection is completed, the winu is updated to show the change [figure 4(b)]. Note that the technician is not forced to bind parameters when establishing a procedure tree node — this activity is triggered by request.

Conditions are a special part of the procedure tree, as distinct from nodes. Adding a condition [figure 5(a)] brings up an instance of the natural language editor. The discourse manager establishes a dynamic lexicon for this instance of the editor that includes (a) all parameters of the test structure itself (in this case MUX, NODE, and CHAN), (b) any local variables created as part of the test structure (in this case none), and (c) any output parameters of tests prior to the condition on the procedure tree (in this case ERRORTYPE and EXCEPTION-TYPE, which are output parameters of the busy_channel measurement). The dynamic lexicon is combined with the static lexicon for expressing conditions (in this case 'the' and '<-'), to produce a word prediction menu, as in figure 5(b). When the condition is correctly expressed, a "thumbs up" icon in the editor window indicates an acceptable entry, as in figure 5(c).

The tree is syntactically valid at any point in the entry process. Note, for instance, that a condition cannot be added unless there are already action nodes to be controlled. Figure 6 shows a fragment of the final tree. ASTEK permits the tree structures to be scrolled in either direction, or panned with a panning cursor.

Other Editors - Other editors currently in use are a menu editor, allowing the selection of one of a set of pre-defined objects from a menu by pointing with the mouse; a list editor, assisting in the creation of a set of objects; and a general text entry editor. The menu editor is used with the parameter binding winu [figure 4], and the text editor is used for the purpose slot [figure 1(c)].

Reference in ASTEK

Two modes of object reference are employed in ASTEK, **direct reference**, using a mouse to point at a visual representation of an object, and **descriptive reference**, using a phrase that describes an object. In keeping with the design principles of ASTEK, direct reference is employed wherever possible and several forms of direct reference are employed:

- from a menu of object representations
- from a visible object stack
- from a graphical depiction of a model or a procedure

The use of descriptive reference is employed whenever the objects to be referred to are not readily at hand. This situation exists in ASTEK only when employing the natural language editor. To aid in descriptive reference and to support the design principle of constraining input to be correct, the lexicon is pre-loaded with recognizable names of tests and measurements in the local context. This *dynamic lexicon* works well for a small collection of objects that are within the local context and minimizes the cognitive burden of searching through the users

memory or even a long menu. For instance, figure 5(b) shows use of a dynamic lexicon of appropriate variable names in defining a branching condition for the procedure tree of the channel_reset test structure. It is not possible to constrain the nodes of a model so readily in troubleshooting systems, however. For references to elements of the model the user is allowed to utilize partial and pronominal descriptions such as "the thing connected to the terminal" to refer to a line driver. The user's description is translated into a feature structure representation and unified with a full description of each of the possible referents. If more than one object unifies with the description the user is asked to select from a menu containing a system-generated description of each of the potential referents.

As measurements are often performed more than once in troubleshooting procedures, we are planning to extend the allowed referring expression for a measurement within a procedure to include such notions as the first, last or n-th invocation of a measurement. We refer to these as *dynamic cues* because editing the steps of a procedure can change which cue is appropriate for a give step. These cues would create features in the description that act as guides to the disambiguation process rather than representing characteristics of the referent.

It is important in the support of object referencing for ASTEK to have the capability to generate unique and recognizable object descriptions. It is critical that these descriptions be concise, especially with the use of menus. Ideally the user should be able to learn to express an object reference to the system by mimicking the system generated object descriptions. If ASTEK utilizes dynamic cues, however, the system can generate descriptions of objects that may not be valid when the user attempts to mimic, thus the ideal cannot be fully realized. A *dynamic object description*, one generated using dynamic cues, is only valid as long as the order of the procedure relative to the objects disambiguated by the dynamic cue remains static.

Reference Scoping

The number of objects in just a modest knowledge base is too large to sort through in order to make a simple reference. ASTEK therefore constrains any menus of objects that can be chosen in a reference activity to those that would be potentially meaningful in the local context. In the natural language editor this is implemented through the use of the dynamic lexicon.

The Recommend Function

To combat the feeling of open-endedness felt by the user, the desire to know whether the knowledge being entered is enough to add up to a complete set of knowledge, the user can request that ASTEK identify a structure to work on next. The recommendation is based on several pre-defined criterion:

- a structure that is complete but has erroneous information in it
- a structure that has been referenced by another structure but does not yet exist
- a structure that has been edited but is not yet complete

The recommendation of an incomplete structure is prioritized last with the intuition that if a structure is left incomplete but the user is still asking for assistance,

then the user must have had some reason for leaving the structure unfinished. This capability will aid the user in creating a knowledge base that is complete and correct (to some level).

Figure 8(a) shows the technician requesting a recommendation after completing the channel_reset test. ASTEK determines that the test contains a reference to a **channel_verify** measurement that has not been defined, so the winu for editing this knowledge structure is presented [figure 8(b)].

The recommend function does aid in reducing the number of incomplete or incorrect knowledge structures in the knowledge base but it is not a comprehensive theory in guiding the user to create a complete and correct knowledge base. We leave this hard problem for future efforts.

CONCLUSIONS

The path that we have taken in knowledge acquisition tools was to start with a simple natural language solution and extend it where the natural language paradigm failed to meet our needs. This has led us in the direction of a form and editor paradigm of knowledge transfer. This paradigm better supports utterance planning, fragmented articulation and variations in the medium of articulation. We have developed a system that looks much like Musen, et al's OPAL, except that form entries themselves may be complex structures or references to complex structures. It may be fair to ask whether natural language still plays an important role in ASTEK. We feel that natural language is, indeed, important in the following ways. At the level of small knowledge fragments, natural language grammars support the medium of articulation when preferred. At a deeper level, ASTEK was designed with the realization in mind that a knowledge acquisition is an extended dialogue between a human expert and the computer. Drawing on work related to the needs of extended dialogues, we have added features, such as object reference and a recommendation function, that make knowledge acquisition more than the isolated editing of templates. The discourse tools help individual acts of articulation be performed as desired while bringing together the individually articulated knowledge fragments into a single, coherent knowledge base as the acquisition task reaches completion.

REFERENCES

[Alexander1986a] Alexander, J.H., Freiling, M.J., Shulman, S., and Staley, J.; "Knowledge Level Engineering: Ontological Analysis;" in *1986 National Conference on Artificial Intelligence*; AAAI; Philadelphia, PA; August 11-14, 1986

[Bobrow1986a] Bobrow, D.G., Kaplan, R.M., Kay, M., Norman, D.A., Thompson, H., and Winograd, T.; "GUS, A Frame-Driven Dialog System;" pp. 595-604, in *Readings in Natural Language Processing*; ed. B.J. Grosz, K. Sparck Jones, B.L. Webber; Morgan Kaufmann Publishers, Inc.; Los Altos, CA; 1986

[Freiling1984a] Freiling, M.J. and Alexander, J.H.; "Diagrams and Grammars: Tools for the Mass Production of Expert Systems;" in *First Conference on Artificial Intelligence Applications*; IEEE Computer Society; Denver, Colorado; December, 1984

[Freiling1984b] Freiling, M.J., Alexander, J.H., Feucht, D., and Stubbs, D.; *GLIB -- A Language for Describing the Behavior of Electronic Devices*; Applied Research Technical Report CR-84-12; Tektronix, Inc.; Beaverton, OR; April 6, 1984

[Freiling1985a] Freiling, M.J., Alexander, J.H., Messick, S.L., Rehfuss, S., and Shulman, S.; "Starting a Knowledge Engineering Project -- A Step-by-Step Approach;" *AI. Magazine*; vol. 6, no. 3, Fall, 1985

[Freiling1986a] Freiling, M.J., Alexander, J.H., Messick, S.L., Rehfuss, S., and Shulman, S.; "The Ontological Structure of a Troubleshooting System for Electronic Instruments;" in *Applications of Artificial Intelligence in Engineering Problems*; ed. D. Sriram and R. Adey; Springer-Verlag; New York, NY; 1986

[Grosz1986a] Grosz, B.J. and Sidner, C.L.; "Attentions, Intentions, and the Structure of Discourse;" *Computational Linguistics*; vol. 12, no. 3, July-September, 1986

[Gruber1987a] Gruber, T.; *Acquiring Strategic Knowledge from Experts*; University of Massachusetts Experimental Knowledge Systems Laboratory; Amherst, Massachusetts; February, 1987

[Guindon1987a] Guindon, R., K.Shuldberg,, and J.Conner,; "Grammatical and Ungrammatical Structures in User-Advisor Dialogues: Evidence for Sufficiency of Restricted Languages in Natural Language Interfaces to Advisory Systems;" in *25th Annual Meeting of the Association for Computational Linguistics*; ACL; Stanford, California; July 6-9, 1987

[Kay1985a] Kay, M.; "Unification in Grammar;" in *Natural Language Understanding and Logic Programming*; ed. V. Dahl and P. Saint-Dizier; North-Holland; New York, NY; 1985

[Kittoa] Kitto, C.M. and Boose, J.H.; "Heuristics for Expertise Transfer: The Automatic Management of Complex Knowledge Acquisition Dialogs;" in *1986 Workshop on Knowledge Acquisition for Knowledge-Based Systems* ; Banff, Canada; November 3-7, 1986

[Messick1986a] Messick, S.L.; *KIERYA: A Tool for Building Natural Language Interfaces*; Applied Research Technical Report CR-86-65; Tektronix, Inc.; Beaverton, OR; 1986

[Musena] Musen, M.A., Fagan, L.M., Combs, D.M., and Shortliffe, E.H.; "Using a Domain Model to Drive An Interactive Knowledge Editing Tool;" in *1986 Workshop on Knowledge Acquisition for Knowledge-Based Systems* ; Banff, Canada; November 3-7, 1986

[Normana] Norman, D.A. and Rumelhart, D.E.; *Explorations in Cognition*; W.H. Freeman and Company; San Francisco, CA; 1975

[Phillips1985a] Phillips, B., Freiling, M.J., Alexander, J.H., Messick, S.L., Rehfuss, S., and Nicholl, S.; "An Eclectic Approach to Building Natural Language Interfaces;" in *23rd Annual Meeting of the Association for Computational Linguistics*; Chicago, IL; July 8-12, 1985

[Phillips1985b] Phillips, B., Messick, S.L., Freiling, M.J., and Alexander, J.H.; *INKA: The INGLISH Knowledge Acquisition Interface for Electronic Instrument Troubleshooting Systems*; Applied Research Technical Report CR-85-04; Tektronix, Inc.; Beaverton, OR; January 7, 1985

[Rehfuss1985a] Rehfuss, S., Alexander, J.H., Freiling, M.J., Messick, S.L., and Shulman, S.J.; *A Troubleshooting Assistant for the Tektronix 2236 Oscilloscope*; Applied Research Technical Report CR-85-34; Tektronix, Inc.; Beaverton, OR; September 25, 1985

APPENDIX -- ANNOTATED ASTEK SESSION

Although ASTEK can be tailored to the acquisition of most forms of structured knowledge, we will illustrate its features with respect to the task of capturing knowledge about troubleshooting a data communications multiplexer. A multiplexer accepts communication signals from several channels, and combines them into one high-speed signal for transmission across a communication line. At the other end, a companion multiplexer accepts the high-speed signal and breaks it

up into its component channels for further transmission along separate communication lines.

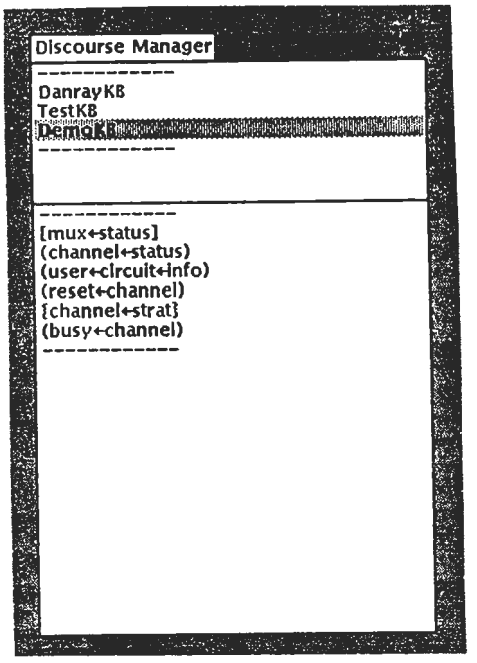
The example we will deal with involves a test and repair procedure for resetting some particular channel card on one of a pair of multiplexers. Although some knowledge-based troubleshooting systems distinguish between diagnosis and repair tasks, our experience is that the two are inextricably intertwined. The channel reset procedure will repair several simple multiplexer faults, but its failure (and the way in which the reset procedure fails) yields information useful for further diagnostic attempts.

The test we define is performed on a particular multiplexer pair, one node of the pair (either the "local" or the "remote" node), and one channel that the multiplexer. The same procedure can be invoked to troubleshoot channel 7 of the local multiplexer, channel 4 of the remote multeplexer, etc. Thus we need to parameterize the test with MUX, NODE, and CHANNEL parameters. ASTEK supports the binding of parameters as tests are constructed. The HIPE inference engine distinguishes four levels of troubleshooting activity, **strategies, tests, measurements, and actions**. Strategies guide the overall localization process. Tests are structured procedures determining the status of specific objects in the system. Measurements are procedures for acquiring values in testing an object, and actions are the atomic units of behavior, which involve affecting the object under test.

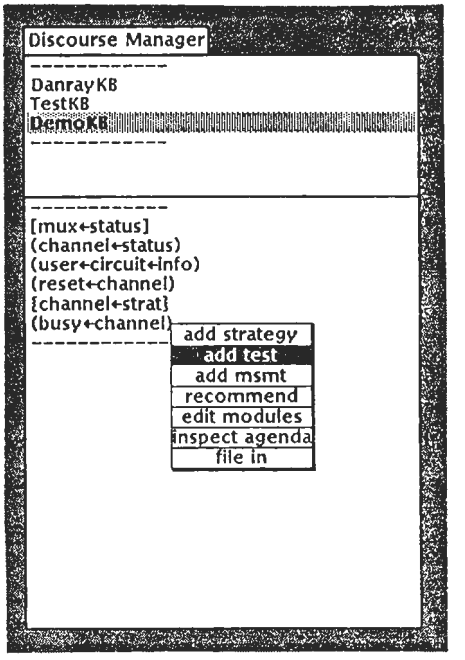
The reset procedure is quite simple. First the channel is "busied out" to establish a known state. Then the reset measurement is executed, and finally a verification test is performed to establish that the channel has indeed been reset to an operational state. A separate interpretation section of the test knowledge structure describes what diagnostic conclusions may be drawn from different results. If the procedure succeeds, the problem is believed to be fixed. If the reset procedure completes, but the verify measurement fails, the problem is believed to be with the channel hardware itself. On the other hand, if a timeout occurs in attempting to busy out the channel, the problem can only be localized to the multiplexer.

Figure 1 -- The object reference stack is displayed in a pane of the Discourse Manager window (a). The user has explicitly requested that a test be created (b) and has begun describing the test (c). In the winu for the test the user has selected the purpose slot. A general text editor has been instantiated in the lower pane of the winu (c) to allow the user to describe the purpose for the test.

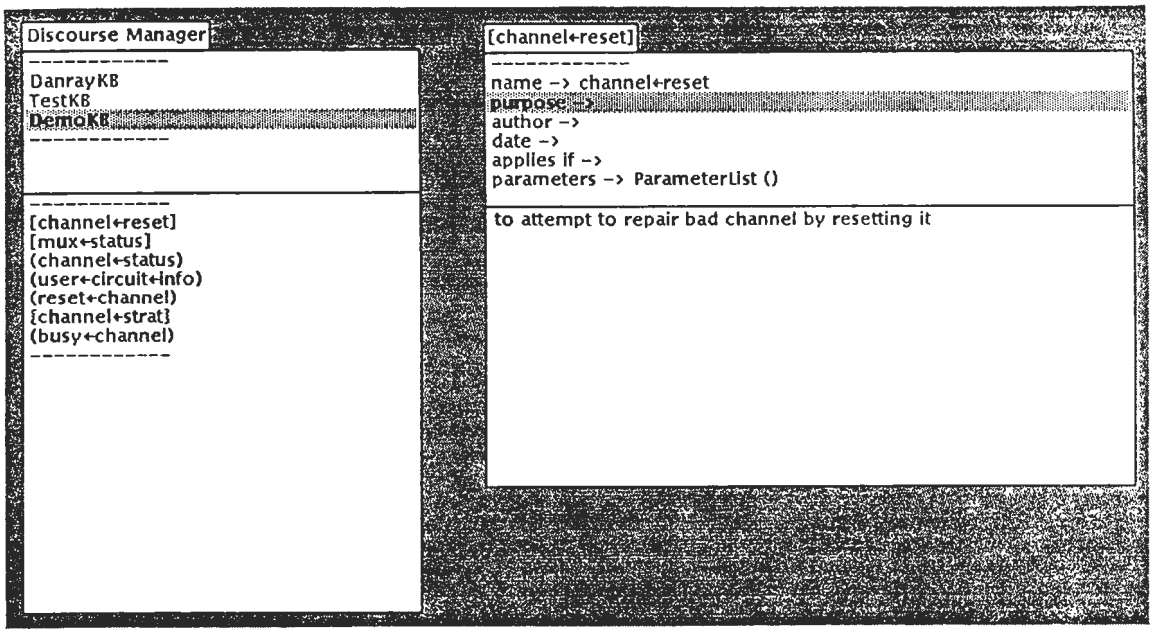
Legend: {a strategy} [a test] (a measurement)



(a)

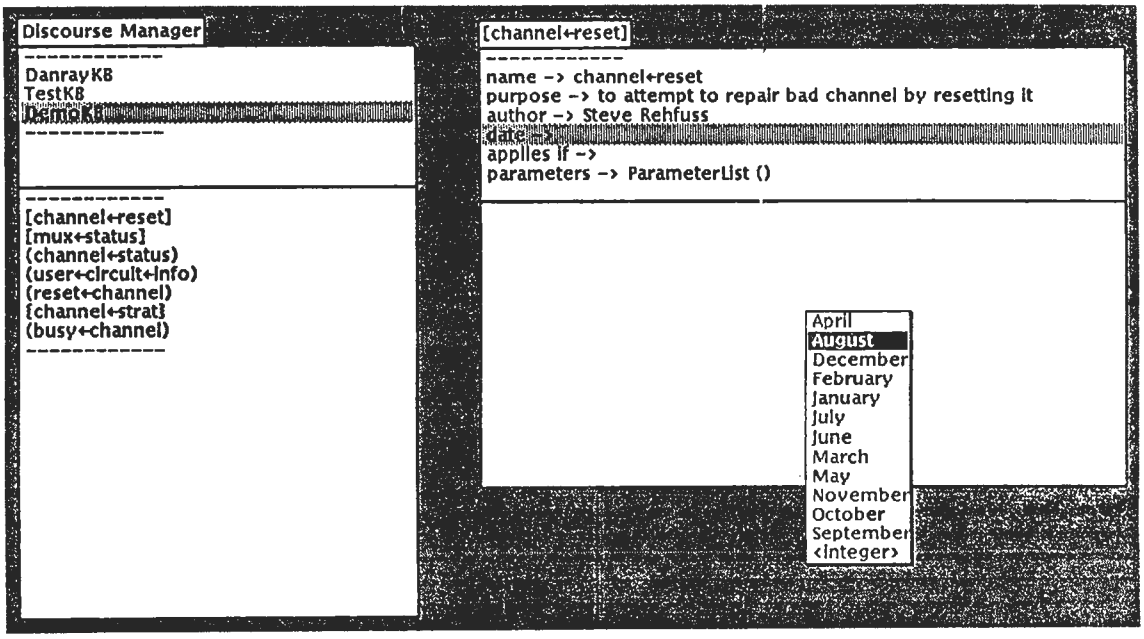


(b)

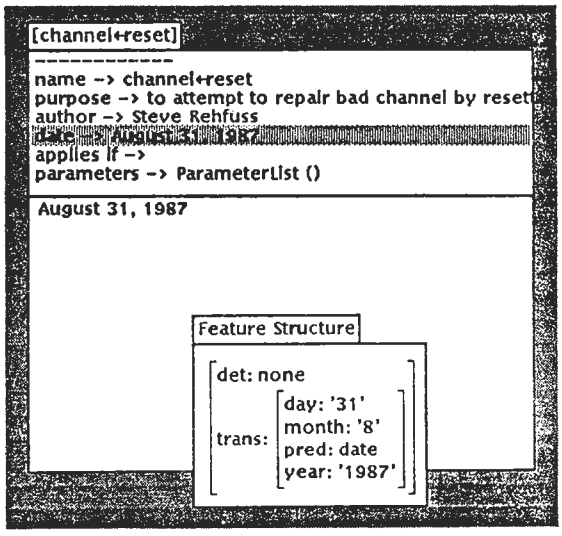


(c)

Figure 2 -- For the date slot, a natural language editor has been invoked. The user has asked the editor to predict what words will be valid (a). In (b) the parse has been accepted and a window has been opened which displays the feature structure representing the translation for the sentence.

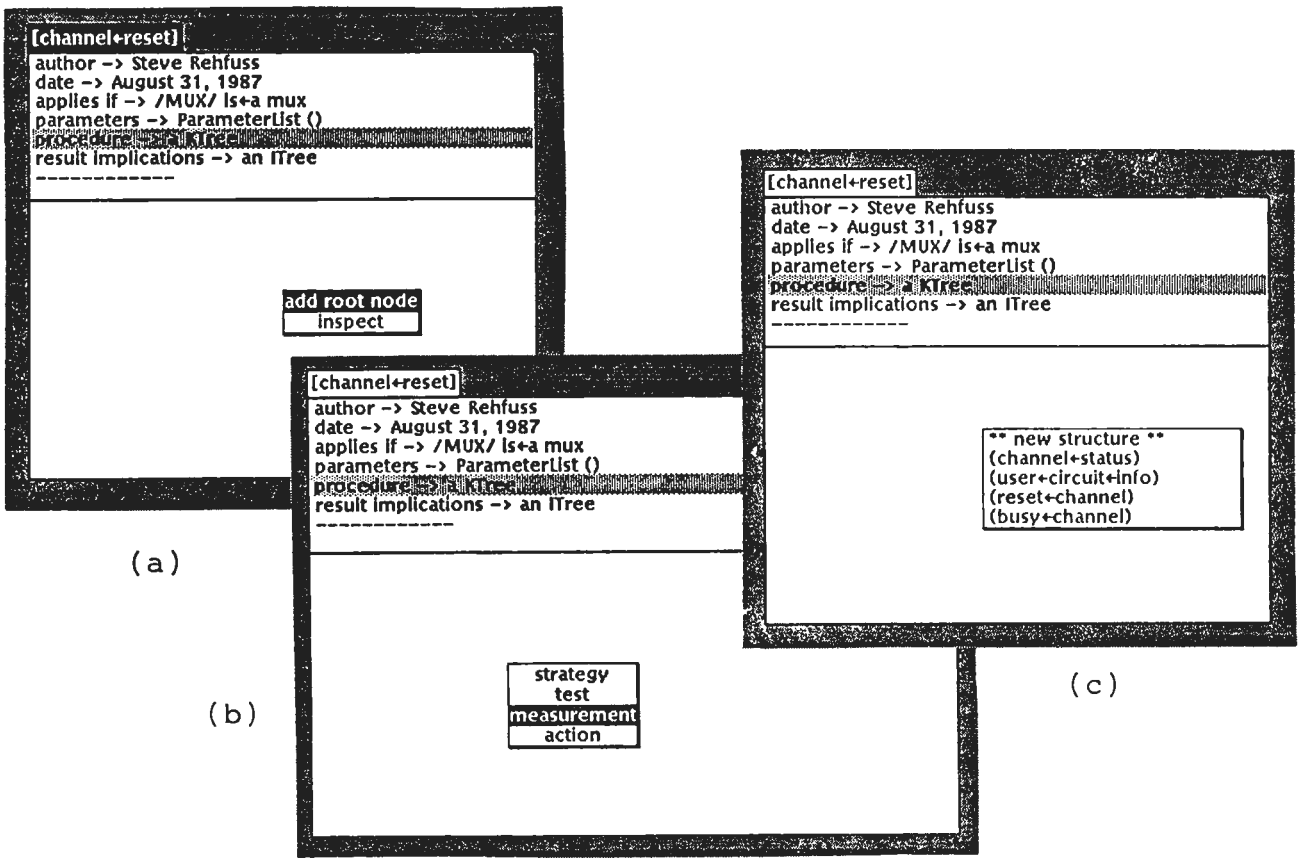


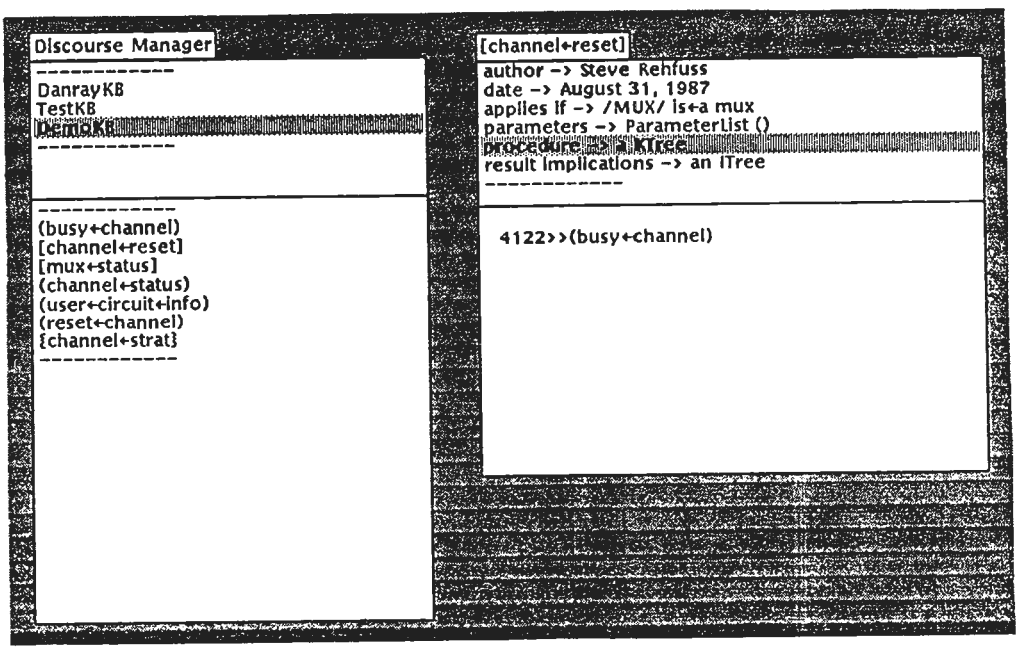
(a)



(b)

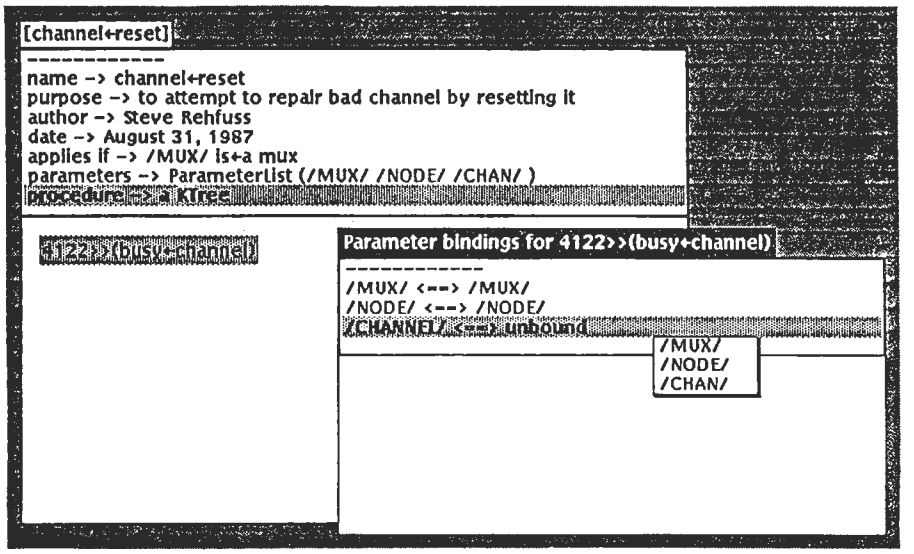
Figure 3 -- A tree editor has been invoked for the procedure slot. The user has requested that a step be added (a,b) to the *channel_reset* procedure. An existing measurement may be referred to or a new measurement may be created. The existing measurements that may be chosen from are displayed in a menu (c), ordered by recency of reference (most recently referenced at the top). The addition of *busy_channel* to the procedure causes the measurement to be moved to the top of the global reference stack (d).



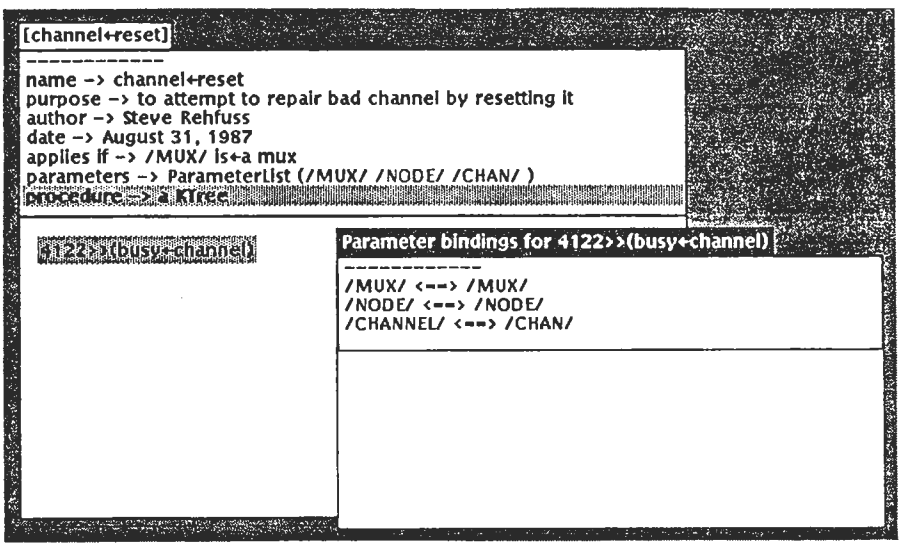


(d) 11-15

Figure 4 -- The *busy_channel* measurement has parameters defined allowing it to be applied to a specific mux, node and channel. These parameters must be bound to local variables in the *channel_reset* test procedure. The local variables in this case are the parameters to *channel_reset* (MUX, NODE, and CHAN). The parameters for the measurement are displayed in a binding window. When a parameter is selected, the local variables are displayed in a menu (a). The *CHANNEL* parameter of *busy_channel* is being bound to the *CHAN* parameter of *channel_reset* (b).



(a)



(b)

Figure 5 -- A condition is being added to the test procedure following the *busy_channel* step (a). A request to edit the condition has opened a natural language window. The word prediction menu contains the dynamic lexicon (b). The local variables included in the dynamic lexicon were parameters to *channel_reset* and output parameters from previous steps of the procedure, in this case the *busy_channel* measurement. In (c) the editor is indicating the successful parse of the sentence with the thumbs-up cursor.

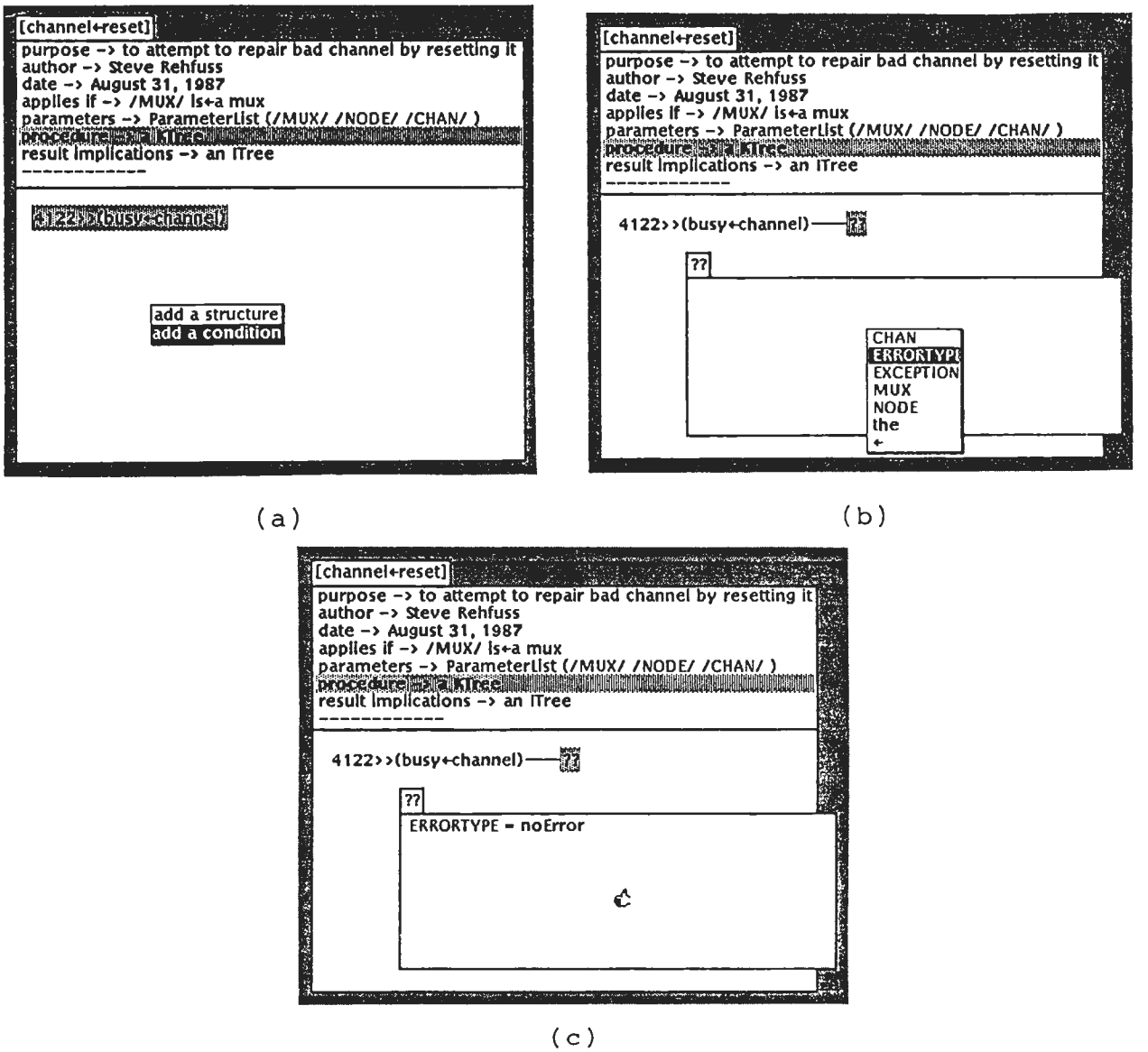


Figure 6 -- A complete test procedure. Visible are measurement invocations, conditional branches and assignments (to create new local variables). The global reference stack reflects the *channel_verify* and *reset_channel* measurements. These structures were added to the top of the reference stack because they were referenced in the *channel_reset* test procedure.

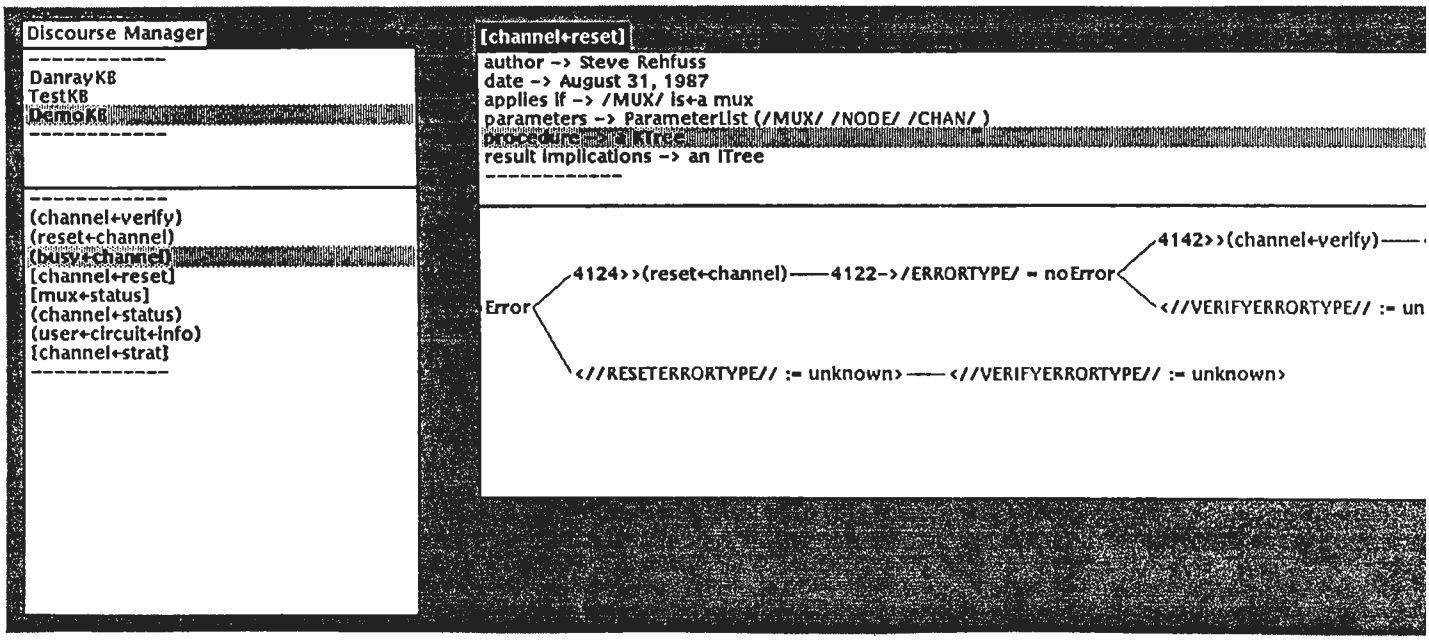


Figure 7 -- A complete implication tree. The local variables representing the results of the test procedure are interpreted according to the implementation procedure. A set of beliefs, diagnostic conclusions, are specified.

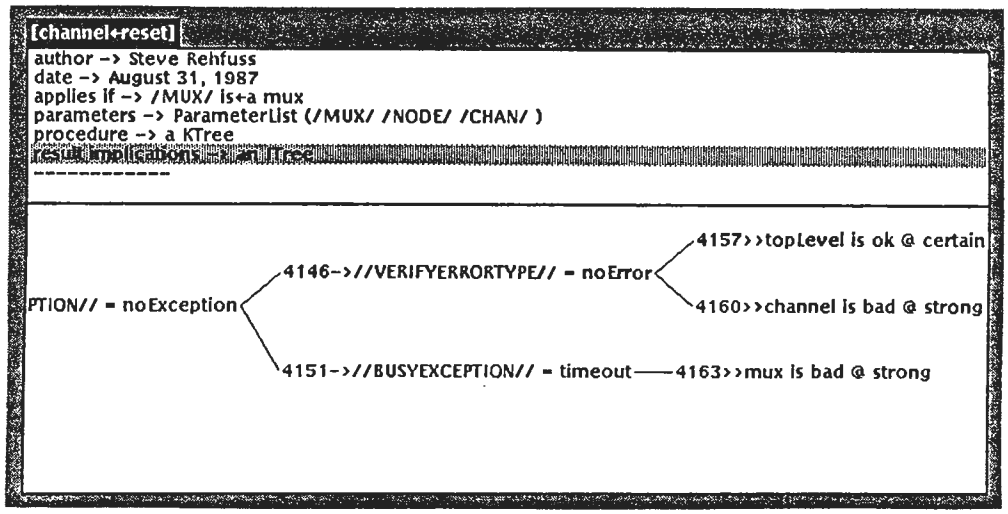
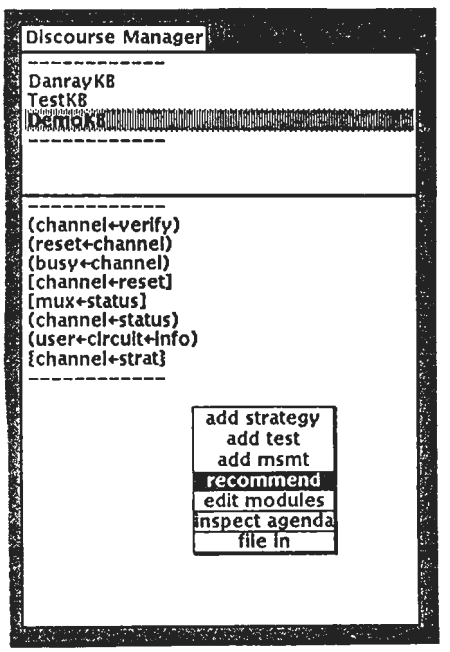
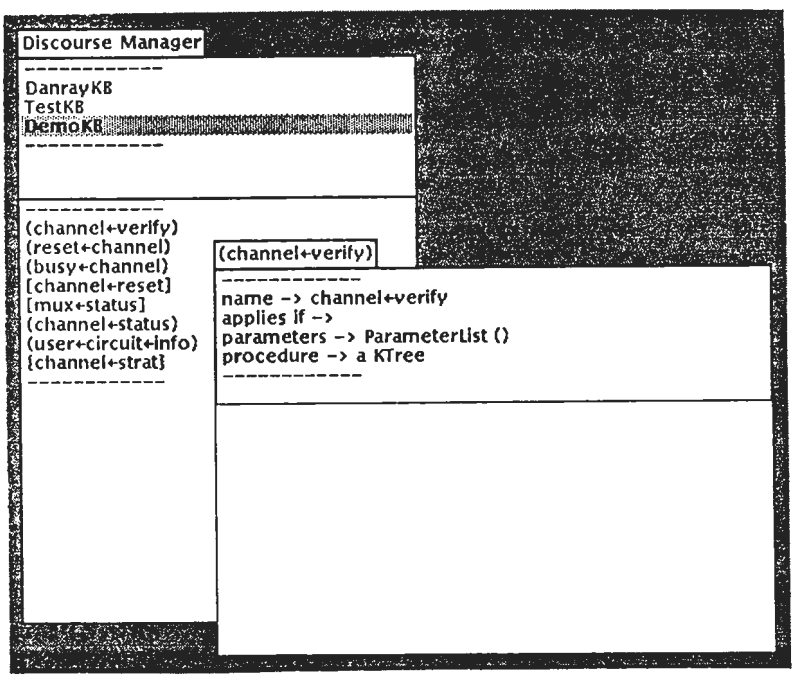


Figure 8 -- The user is asking for ASTEK to recommend a knowledge structure to edit (a). ASTEK suggested the *channel_verify* measurement which has been referenced in the procedure of the *channel_reset* test but has not been described. The user accepted the recommendation causing a winu to be opened for the measurement (b).



(a)



(b)

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

SELECTING KNOWLEDGE ACQUISITION TOOLS AND STRATEGIES BASED ON APPLICATION CHARACTERISTICS

Catherine M. Kitto and John H. Boose

Knowledge Systems Laboratory, Advanced Technology Center
Boeing Computer Services, P.O. Box 24346, M.S. 7L-64,
Seattle, Washington 98124-0346

Submitted, Second Knowledge Acquisition for Knowledge-Based Systems Workshop

ABSTRACT

One of the most troublesome and time-consuming activities in constructing a knowledge-based system is the elicitation and modeling of knowledge from the human expert about the problem domain. A major obstacle is that little guidance is available to the domain expert or knowledge engineer to help with 1) classifying the application task and identifying a problem-solving method, and 2) given the application task characteristics, selecting knowledge acquisition tools and strategies to be applied in creating and refining the knowledge base.

Our objective is to provide automated assistance to a knowledge engineer or domain expert in analyzing the problem domain, classifying the problem tasks and sub-tasks, identifying problem-solving methods, proposing knowledge acquisition tools, and suggesting the use of specific strategies for knowledge acquisition provided in selected tools. We describe such an implementation in the Dialog Manager subsystem of the Aquinas knowledge acquisition workbench. The Dialog Manager provides advice to potential Aquinas users as well as continuing guidance to users who select Aquinas for knowledge base development.

INTRODUCTION

The knowledge acquisition process is considered one of the most challenging steps in the development of an expert system. Choosing the correct problem-solving paradigm and knowledge acquisition tool for the application task is an important step in this process. Attempts to use inappropriate problem-solving methods or knowledge acquisition tools are likely to fail. For example, a tool based on classification problem-solving and intended for applications such as the selection of stereo equipment or diagnosis of engine malfunctions would produce disappointing results if applied to a constructive problem-solving task such as computer configuration. The characteristics of the application task should be carefully evaluated in selecting the knowledge acquisition tools and techniques for building a knowledge-based system.

The Dialog Manager (Kitto and Boose, 1987), the automated guidance facility for the Aquinas knowledge acquisition workbench, provides advice on strategies and tools based on application characteristics. In an initial dialog with the domain expert or knowledge engineer, the Aquinas Dialog Manager helps to identify the type of application task, the problem-solving paradigm, and an appropriate knowledge acquisition tool. These preliminary consultations with prospective Aquinas users are a recent expansion of the role of the Aquinas Dialog Manager. If Aquinas is the knowledge acquisition tool chosen, the Dialog Manger recommends specific

techniques within Aquinas for elicitation and analysis of the expert's knowledge based on the application's attributes.

CHOOSING PROBLEM-SOLVING METHODS FOR APPLICATION TASKS

Classification of Application Tasks

Application tasks can be classified according to various schemes (Stefik et. al.,1982, Hayes-Roth et. al., 1983, Clancey, 1986, Waterman, 1986). These application task categories often include interpretation, prediction, diagnosis, design, planning, monitoring, debugging, repair, instruction, and control (Hayes-Roth, 1983). These generic applications can be divided into two broad groups: those primarily associated with *analysis* (interpretive) tasks, and those concerned with *synthesis* (constructive) activities. These task groups can be further decomposed into classification frameworks of generic tasks and sub-tasks.

Identifying a Problem-solving Method

Given a classification scheme for application tasks, we can use that framework to identify a problem-solving method and suggest the use of knowledge acquisition tools and strategies where they exist. Our initial work is based on a hierarchy of problem tasks proposed by Clancey (1986), but any taxonomy of application tasks could be used as a starting point. Clancey identifies two fundamental problem-solving methods, each of which can be applied to a range of application tasks: *heuristic classification* and *heuristic construction*. Heuristic classification is a common problem-solving method in which concepts in different classification hierarchies are heuristically related using a process of data abstraction, heuristic matching, and solution refinement. This heuristic classification method is usually suitable for problems in diagnosis, repair, catalog selection, or skeletal planning.

In heuristic construction, the problem-solver constructs solutions, either by generating complete solutions or assembling solutions from components while satisfying constraints. This problem-solving method is best suited for synthesis application tasks (design, configuration, planning, scheduling, etc.). Knowledge of how pieces of a solution fit together is essential, and solutions are assembled incrementally. This involves a technique of proposing a solution and reasoning about it, such as the Propose-and-Revise strategy used in the VT expert system for elevator configuration (Marcus, Stout, and McDermott, 1987; Marcus, McDermott, and Wang, 1985).

Chandrasekaran (1985) describes six generic tasks in knowledge-based reasoning which ressemble the problem-solving methods proposed by Clancey. These generic reasoning tasks include classification, state abstraction, knowledge-directed retrieval, object synthesis by plan selection and refinement, hypothesis matching, and assembly of compound hypotheses for abduction. Like Clancey, Chandrasekaran views these generic "tasks" as problem-solving methods which can be combined to perform knowledge-based reasoning for an application task.

DETERMINING KNOWLEDGE ACQUISITION TOOLS AND STRATEGIES

A set of knowledge acquisition strategies has been proposed by Boose and Bradshaw, 1987. Each strategy links certain sub-tasks with appropriate problem-solving methods. Knowledge acquisition strategies include establishing distinctions between alternatives, problem decomposition, combining and propagating information, testing of knowledge, combining multiple sources of knowledge,

incremental expansion of knowledge, and providing process guidance. Similarly, Bylander and Chandrasekaran (1987) suggest that generic problem-solving tasks can be associated with a specific knowledge acquisition methodology. They use classification problem-solving as an illustration.

Selecting Knowledge Acquisition Tools

Knowledge acquisition tools provide links between an application task and the appropriate problem-solving method. If we know the type of application task and have identified an appropriate problem-solving method, we can then select a knowledge acquisition tool which links the application and method if such a tool exists. In figure 1, a hierarchy of application task categories is shown in the lower portion of the figure, with problem-solving methods above. The broken lines depict examples of links supported by existing knowledge acquisition systems. Several knowledge acquisition systems enable diagnostic application tasks to use the heuristic classification problem-solving method. These include MDIS (Antonelli,

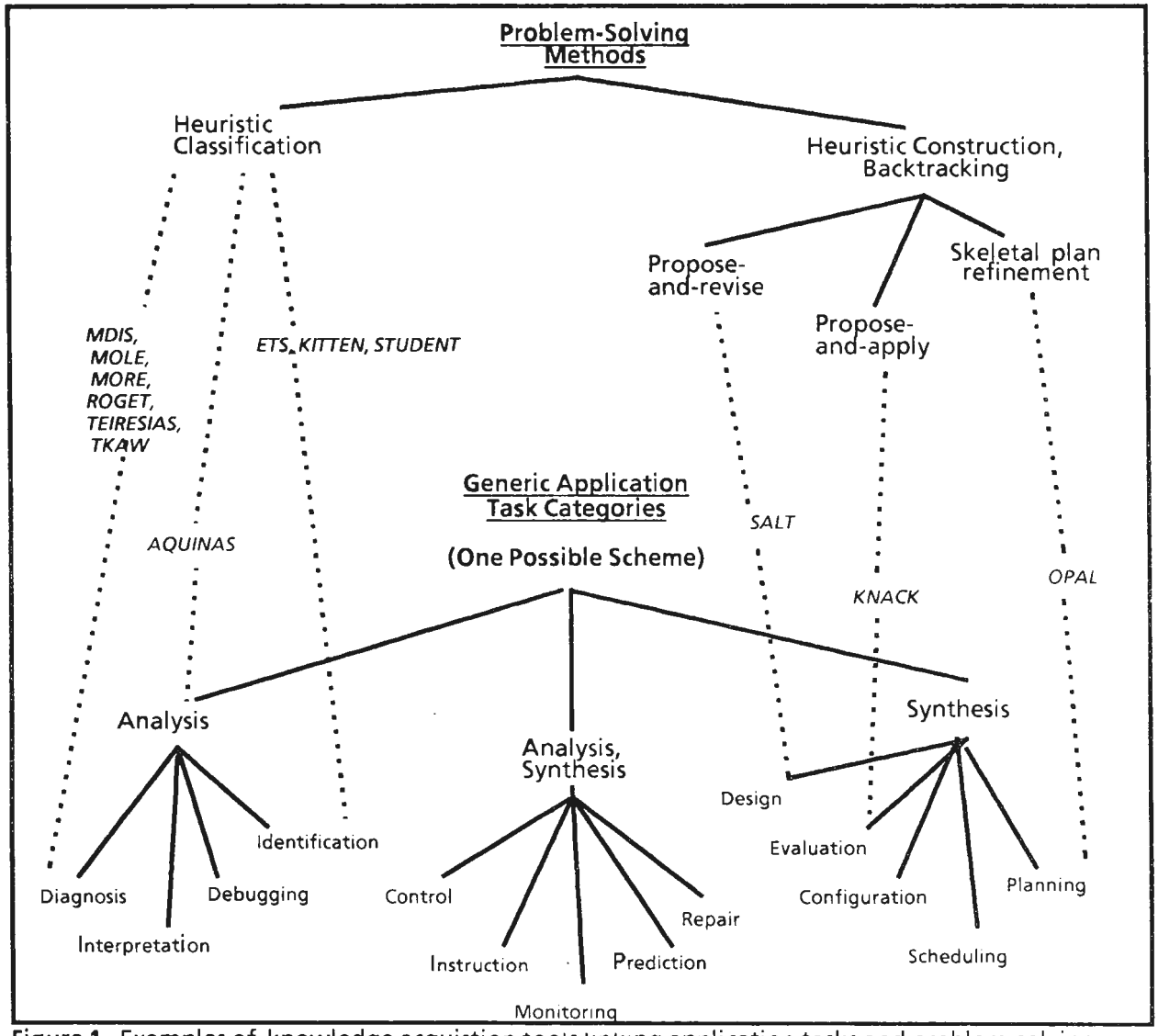


Figure 1. Examples of knowledge acquistion tools linking application tasks and problem-solving methods.

1983), MORE (Kahn, Nowlan and McDermott, 1985), MOLE (Eshelman, Ehret and McDermott, 1987), TEIRESIAS (Davis, 1985), ROGET (Bennett, 1983), and TKAW (Kahn, Breaux, Joseph, and DeKlerk, 1987). These knowledge acquisition systems are tailored for diagnostic applications, but have considerable domain-related power. Likewise, ETS (Boose, 1985; 1986), KITTEN (Shaw and Gaines, 1987), and STUDENT (Gale, 1987) link identification application tasks with heuristic classification capability.

AQUINAS (Boose and Bradshaw, 1987) also supports heuristic classification problem-solving but has more generalized knowledge acquisition capablities (diagnosis, interpretation, debugging, identification) at a cost of some loss of depth in the resulting domain knowledge.

Three knowledge acquisition systems are illustrated for synthesis applications. SALT (Marcus) links design tasks with a propose-and-revise problem-solving approach, while OPAL (Musen, Fagan, Combs and Shortliffe, 1987), constructs and refines plans. KNACK, a knowledge acquisition system for report generation and evaluation (Klinker, Bentolila, Genetet, Grimes and McDermott, 1987; Klinker, Boyd, Genetet, and McDermott, 1987), has characteristics of a synthesis task (assembly of report fragments and evaluation in respect to constraints).

Specific Knowledge Acquisition Strategies for Application Tasks

A knowledge acquisition system must also be able to elicit information to meet the unique problem-solving requirements of an application task category. For example, in *electronic* or *electromechanical diagnosis*, component failure rates, test times, test costs, test complexity, test utility or benefit, and test dependencies are useful to the human expert. According to Simpson and Dowling (1986), a fault isolation strategy for troubleshooting systems such as WRAPLE must be able to incorporate and use information on test complexity, test cost, length of tests, and frequency of component failure.

Another example of specific application task knowledge requirements is described by Hand (1985) for the domain of medical diagnosis. According to Hand, experts in medical diagnosis typically use the following problem-solving strategies:

1). Generate a list of multiple competing hypotheses compatible with the symptoms.

2). Examine the most frequently occurring candidate diseases first.

3). Consider the "utility" of a diagnosis. If there is a high cost associated with failure to diagnose a problem (for example, the patient dies) , then focus on that hypothesis.

4). Consider diagnoses for which treatments are available first.

5) Do not discriminate between alternative diagnoses with the same treatment.

6). Avoid redundant or superfluous questions or tests. Each piece of information requested should play a role in diagnosis.

7). Evaluate tradeoffs between the benefit of obtaining information against its cost and risk.

New knowledge acquisition strategies are needed to capture the medical diagnosis problem-solving knowledge. Strategies should permit elicitation of probabilities of diseases, utility of a correct diagnosis, and the relative cost and risk of obtaining information. Experts should be able to specify the ordering of questions as well as

initial data to be collected at the start of each consultation (for example, patient name, age, sex, complaint, and observations of the physician).

AUTOMATED ASSISTANCE IN KNOWLEDGE ACQUISITION DECISIONS

The selection of knowledge acquisition strategies for particular application tasks has been addressed in the implementation of the *Dialog Manager* (Kitto and Boose, 1987), part of the AQUINAS system (figure 2). The Dialog Manager provides guidance in constructing a knowledge base with the AQUINAS system. A major component of AQUINAS is ETS (Boose, 1985) which elicits, analyzes, and refines knowledge represented in repertory grids. The interviewing methodology in AQUINAS is based on personal construct theory (Kelly, 1955; Gaines and Shaw, 1981; Shaw and Gaines, 1987).

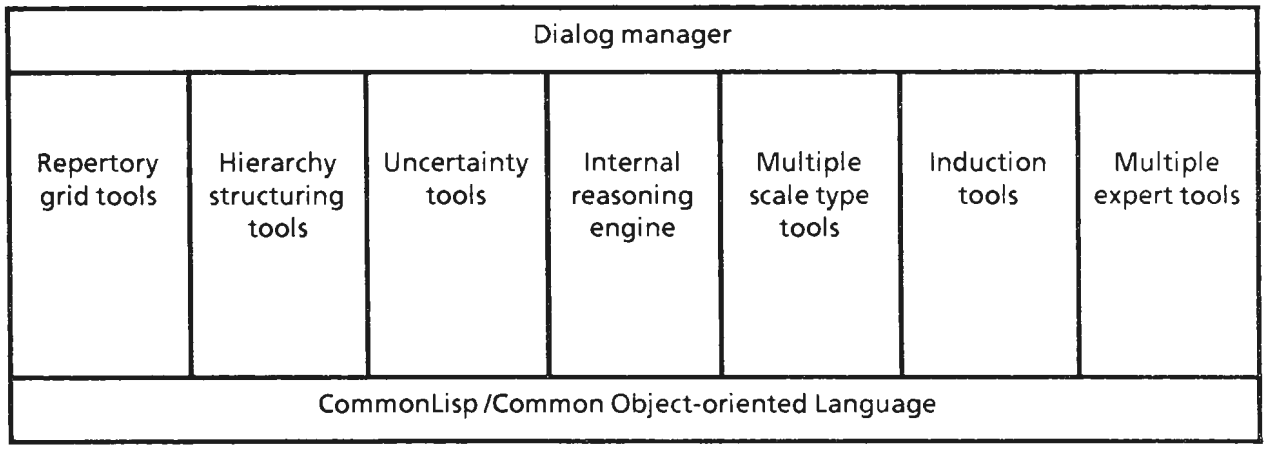


Figure 2. The Dialog Manager provides advice for all tools in the AQUINAS workbench..

The Dialog Manager is a knowledge-based system consisting of heuristics and strategies for knowledge acquisition encoded as rules and procedures. Its domain knowledge concerns the use of AQUINAS and the characteristics of other selected knowledge acquisition tools. It offers a mixed initiative environment with options to set the levels of control, guidance, and prompting. The Dialog Manager examines its set of rules to select the most appropriate knowledge acquisition technique to be applied based upon heuristics which consider the application task, problem-solving method, current state of the knowledge base, user experience, temporal considerations, and user preferences.

Advice in choosing knowledge acquisition strategies is offered at two stages: First, in an initial interview with the domain expert, the Dialog Manger queries the expert about the nature of the application task. Based on the task's characteristics and similarities to types of application tasks with which the Dialog Manager is familiar, the Dialog Manager classifies the application task selects an appropriate problem-solving method for the task and recommends a specific knowledge acquisition tool. Second, for those application tasks for which AQUINAS is the suggested knowledge acquisition system, the Dialog Manager will guide the expert or knowledge engineer in acquiring the necessary problem-solving knowledge. The Dialog Manager considers characteristics of application and problem-solving methods to select appropriate knowledge acquisition strategies within AQUINAS.

Determining Problem-Solving Methods for an Application -- Examples

The Dialog Manager assists in classifying an application into one or more sub-tasks; each sub-task implies the use of a problem-solving method. This step is essential since it will determine the knowledge acquisition tools to be applied. For example, if the task is identification, useful knowledge acquisition techniques include those embodied in tools such as ETS, and heuristic classification would be the method employed. However, for a design or planning task, other methods should be applied such as the Propose-and-Revise method used in the SALT knowledge acquisition tool for constructive systems (Marcus, 1987). The Dialog Manager conducts a preliminary interview with the domain expert or knowledge engineer in the initial stages of expert system development.

```
-- AQUINAS PRELIMINARY CONSULTATION --
Would you like to work on an OLD knowledge base, or create a NEW one?
AQU**NEW
Do you already know the type of problem task or sub-tasks that you are addressing in this
knowledge base?
AQU**NO
--[DM] Analyzing Problem Tasks and Sub-Tasks--
Give a description of the general class of problem you are trying to solve:
AQU**JET ENGINE DIAGNOSTICS
Please give a short description of the problem:
AQU**DIAGNOSES JET ENGINE FAULTS BASED ON EVENTS AND SYMPTOMS
Do you already have a list of pre-enumerated solutions to the problem you are trying to solve?
AQU**YES
Approximately how many possible conclusions or solutions are there?
AQU**100
Can these solutions be organized into a hierarchy of solution classes and solution children?
AQU**YES
What would you say is the maximum number of "solution children " that would occur in any
solution class?
AQU**10
Can you select a subset of the most significant solutions and limit your problem domain to this sub-
set?
AQU**YES
Approximately how many possible conclusions or solutions are there in this sub-set?
AQU**15
--[DM] Your application task should be treated as a DIAGNOSIS sub-task.
An appropriate problem-solving method is HEURISTIC CLASSIFICATION.
```

Figure 3. Preliminary consultation with an expert to determine application task and problem-solving method.

The first example (figure 3) illustrates how the Dialog Manager employs its heuristics for knowledge acquisition to guide an expert in classifying the application and identifying the problem-solving method to construct a generic jet engine diagnostics system.

We show an example of the preliminary dialog between the domain expert and the Dialog Manager concerning the task to diagnose malfunctions in the generic jet engine system. Recommendations from the Dialog Manager are highlighted and preceded by [DM], while Aquinas user entry is in underlined bold type. The Dialog Manager determines that the application is a diagnostic task and that heuristic

classification is the appropriate problem-solving method. Several knowledge acquisition tools for diagnostic applications are suggested, including MOLE, TKAW, and AQUINAS. Figure 4 shows the recommendations of the Dialog Manager.

--[DM]--Knowledge acquisition tools for electromechanical **DIAGNOSTIC** problems include: **MOLE, TKAW,** and **AQUINAS**.

--[DM]--For your application, I recommend using the **AQUINAS** system since it is available at your site.

Figure 4. The Dialog Manager recommends knowledge acquisition tools based on application task and problem-solving method.

The preliminary consultation is intended to analyze all types of application tasks, even those which are unsuited for the Aquinas problem-solving paradigm. Figure 5 shows an example consultation with an expert in office space configuration.

--AQUINAS PRELIMINARY CONSULTATION--
Would you like to work on an OLD knowledge base, or create a NEW one?
AQU**NEW
Do you already know the type of problem task or sub-tasks that you are addressing in this knowledge base?
AQU**NO
--[DM] Analyzing Problem Tasks and Sub-Tasks--
Give a description of the general class of problem you are trying to solve:
AQU**CONFIGURATION
Please give a short description of the problem:
AQU**CONFIGURING SPACE FOR OFFICES
Do you already have a list of pre-enumerated solutions to the problem you are trying to solve?
AQU**NO
Can you generate or construct a large number of possible solutions from components?
AQU**YES
What are the most important items to be configured? (Enter separated by commas)
AQU**DESKS, TABLES, BOOKCASES, CHAIRS
Are there constraints which apply to the number of DESKS, TABLES, BOOKCASES, or CHAIRS per OFFICE?
AQU**YES
Does the application involve spatial or temporal constraints?
AQU**YES
Are there possible fixes to be applied if a constraint is violated?
AQU**YES
Can these fixes cause violations of other constraints?
AQU**YES
•
•
•
--[DM] Your application task should be treated as a **CONFIGURATION** sub-task.
The appropriate problem-solving method is **NOT HEURISTIC CLASSIFICATION**.
We do **NOT** recommend the use of **AQUINAS** for this application. Better results may be obtained with the use of a constructive problem-solving tool such as **SALT**.

Figure 5. In this interview, it is determined that the application task is not appropriate for the heuristic classification problem-solving method or AQUINAS.

The Dialog Manager conducts a preliminary interview and concludes that the application requires a problem-solving method beyond the scope of the heuristic classification method embodied in knowledge based systems produced by Aquinas. Instead, the Dialog Manager suggests that the expert consider a knowledge acquisition tool designed for constructive problem-solving such as SALT or KNACK. Currently, no specific tool is recommended for applications outside the scope of Aquinas.

Like the ROGET system (Bennett, 1983), the expert can select similar tasks and characteristics from menus of tasks already categorized.

Dialog Manager Assistance in Choosing Knowledge Acquisition Strategies

In the examples below, Aquinas and the Dialog Manager were used to develop the knowledge base for a generic jet engine diagnostic system. Diagnostic problems have unique requirements for problem-solving. The Aquinas Dialog Manager will suggest different knowledge acquisition strategies for a diagnostic task than for the more general task of identification. For example, in electronic diagnosis, an expert will assess the relative probabilities of occurrence of various malfunctions in determining how to focus the diagnostic process. This estimate of frequency may take the form of "mean time between failure" data, or it may be a subjective judgement based upon the expert's past experience. Likewise, in medical diagnosis, the physician will consider the frequency of various disorders in selecting hypotheses for consideration, seeking evidence for the most frequently occurring diseases first (Hand).

In order to replicate this approach to problem-solving, a knowledge acquisition system must capture knowledge about frequencies of possible problem solutions for diagnostic problems. In figure 6, the Dialog Manager applies a heuristic based upon domain characteristics of diagnostic problems and recommends that the expert enter the relative frequencies of malfunctions. These estimates will be used as prior probabilities in the reasoning process in the completed expert system.

---[DM]--- I recommend that you
EDIT FILL.IN. SOLUTION WEIGHTS
since the application task is diagnosis and since you have already entered and rated a set of solution candidates, and since component failure rates (prior probabilities) are important in diagnostic problem-solving.

AQU**EDIT FILL.IN.SOLUTION WEIGHTS (*expert agrees*)

Please estimate the relative frequencies (weights) of each of the following solution classes, entering values from 0 to 1. For electromechanical diagnosis, weights may be estimates of component failure rate or MTBF's.

AQU**FUEL.SYSTEM.MALFUNCTION ** .35
AQU**BLEED.VALVE.CONTROL.UNIT.MALFUNCTION** .45
AQU**COMPRESSOR.STALL ** .05
⋮

Figure 6. The Dialog Manager suggests that solution weights (relative frequencies) be entered for a diagnostic problem.

Another knowledge acquisition strategy which is appropriate for diagnostic applications is to elicit information about the relative "costs" of obtaining information. Costs may be a subjective estimate and may include such aspects as the dollar cost of a test, length of test, level of skill of testing personnel, and risk of the test. This information is often used in diagnostic problem-solving to determine whether to schedule a specific diagnostic test. Figure 7 depicts the recommendation of the Aquinas Dialog Manager.

```
---[DM]--- I recommend that you
    TEST CONFIGURE DIAGNOSTIC.PROBLEMS TRAIT.COSTS
since the application task is diagnosis and since cost of obtaining information is
important in diagnostic problem-solving. "Cost" may include estimates of risk, cost in
dollars, cost in time, cost in resources, etc.

AQU** TEST CONFIGURE DIAGNOSTIC.PROBLEMS TRAIT.COSTS        (expert agrees)

Please estimate the relative costs of obtaining information on each problem-solving trait
on a scale of 1 to 5 where 1 is inexpensive and 5 is costly. "Cost" should represent dollars,
time, and risk. Enter CHANGE to specify a different scale.

AQU** PART.POWER.TRIM.TEST **3
AQU** ACCELERATION TO SOME POWER SETTING -- IN.FLIGHT ** 5
AQU** ACCLERATION TO SOME POWER SETTING --GROUND TEST ** 4
AQU** IDLE.TRIM.TEST ** 2
AQU** CIRCUIT CONTINUITY TEST ** 1
       :
       :
```

Figure 7. The Dialog Manager recommends that trait costs (cost in time, money, risk to obtain information) be entered for a diagnostic problem.

The examples (figures 6 and 7) show the Dialog Manager recommending specific knowledge acquisition strategies and operations for a diagnostic application. These commands would not be suggested for some other type of application.

RESULTS

This approach has been used successfully to guide a number of knowledge-based system development efforts using Aquinas. While still in its infancy, we feel that our experience with the Dialog Manager demonstrates that the knowledge-based system development process can be facilitated by providing automated assistance with problem task categorization, selection of a problem-solving method for a given task, and recommendations of appropriate knowledge acquisition tools and strategies.

ISSUES AND FUTURE WORK

The implementation of the Dialog Manager and the experience of experts and knowledge engineers who have used Aquinas and the Dialog Manager to develop prototype expert systems have raised several issues. We are only beginning to understand the techniques currently used by experts or knowledge engineers using an automated system such as Aquinas, their underlying heuristics, and whether there are more effective strategies for knowledge acquisition. We expect to learn

more about knowledge acquisition strategies for specific application types as more experts and knowledge engineers have the opportunity to use Aquinas with the Dialog Manager.

We also anticipate that the requirements for guidance from the Dialog Manager will increase as the capability is added to Aquinas to acquire knowledge for problem domains which do not fit the identification task--heuristic classification paradigm. We hope to extend Aquinas to handle knowledge acquisition for constructive problem-solving such as in the SALT system and to represent domain models of qualitative causal relations in acquiring diagnostic knowledge as in the MORE or MOLE systems.

One issue is whether it is possible for a single tool to have knowledge of many problem domains in order to classify an application task, identify its problem-solving method, and suggest a tool or class of tools. We already recognize that advice on specific knowledge acquisition strategies must be closely associated with the tool itself. It is doubtful that a single advisory system could direct and monitor knowledge base development under several knowledge acquistion tools, particularly where the tool relies on built in domain knowledge.

It is also apparent that knowledge acquisition tools do not currently exist for certain application types or problem-solving methods. In addition, it is possible for a complex application to require several problem-solving methods to resolve the total problem. Most current knowledge acquisition tools support only one problem-solving method. We expect that this situation will be improved as more domain-specific or more generalized knowledge acquisition tools are created.

ACKNOWLEDGEMENTS

The authors wish to thank Miroslav Benda, Jeff Bradshaw, Jackson Brown, Ted Kitzmiller, Georg Klinker, Art Nagai, Sandra Marcus, Dave Shema, Sasha Smith, Lisle Tinglof-Boose, and Bruce Wilson for their contributions and support. AQUINAS and the Dialog Manager were developed at the Knowledge Systems Laboratory at the Boeing Advanced Technology Center of Boeing Computer Services in Seattle, Washington.

REFERENCES

Antonelli, D., "The Application of Artificial Intelligence to a Maintenance and Diagnostic Information System (MDIS)" in the *Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance,* Boulder, Co., 1983.

Bennett, J.S., "ROGET: A Knowledge-based Consultant for Acquiring the Conceptual Structure of an Expert System", *Report No. HPP-83-24, Computer Sciences Department, Stanford University*, 1983.

Boose, J. H., "A Knowledge Acquisition Program for Expert Systems Based On Personal Construct Psychology", *International Journal of Man-Machine Studies*, **23**, 1985.

Boose, J.H., *Expertise Transfer for Expert System Design*, New York: Elsevier, 1986.

Boose, J.H. and Bradshaw, J.M., "Expertise Transfer and Complex Problems: Using AQUINAS as a Knowledge Acquisition Workbench for Expert Systems", *Special*

Issue of the International Journal of Man-Machine Studies on the AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, to be published, 1987.

Boose, J., and Bradshaw, J. "Missing Links: Bridging Problem-Solving Tasks and Methods with Knowledge Acquisition Strategies and Tools", *Boeing Computer Services Technical Report*, to be published, 1987.

Bylander, T., and Chandrasekaran, B., "Generic Tasks for Knowledge Based Reasoning: the 'Right' Level of Abstraction for Knowledge Acquisition", *Special Issue of the International Journal of Man-Machine Studies on the AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,* to be published, 1987.

Chandrasekaran, B., "Generic Tasks in Knowledge Based Reasoning: Characterizing and Designing Expert Systems at the 'Right' Level of Abstraction", *Proceedings of the Second Conference on Artificial Intelligence Applications, Miami Beach Florida,* pp. 294-300, New York: IEEE Computer Society, 1985.

Clancey, W.J., "Heuristic Classification", in *Knowledge Based Problem Solving*, Kowalik, J.S. (ed.), pp. 1-67, Edgewood Cliffs, N.J.: Prentice Hall, 1986.

Davis, R., "Interactive Transfer of Expertise", in Buchanan, B. and Shortliffe, E. (eds.) *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Reading, Mass: Addison-Wesley, 1985.

Eshelman, L., Ehret, D., McDermott, J., and Tan, M., "MOLE: A Tenacious Knowledge Acquisition Tool", *Special Issue of the International Journal of Man-Machine Studies on the AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,* to be published, 1987.

Gaines, B.R., and Shaw, M.L.G., "New Directions in the Analysis and Interactive Elicitation of Personal Construct Systems", in M.L.G. Shaw (ed), *Recent Advances in Personal Construct Technology*, New York: Academic Press, 1981.

Gale, W.A., "Knowledge Based Knowledge Acquisition for a Statistical Consulting System", *Special Issue of the International Journal of Man-Machine Studies on the AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,* to be published, 1987.

Hand, D. J., *Artificial Intelligence and Psychiatry*, Cambridge: Cambridge University Press, 1985.

Hayes-Roth, F., Waterman, D., and Lenat, D.B., *Building Expert Systems*, Reading, Mass: Addison-Wesley, 1983.

Kahn, G., Nowlan, S., and McDermott, J., "MORE: An Intelligent Knowledge Acquisition Tool", in the *Proceedings of the Ninth Joint Conference on Artificial Intelligence*, Los Angeles, California, August, 1985.

Kahn, G.S., Breaux, E.H., Joseph, R.L., and DeKlerk, P., "An Intelligent Mixed Initiative Workbench for Knowledge Acquisition", *Special Issue of the International Journal of Man-Machine Studies on the AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,* to be published, 1987.

Kelly, G.A., *The Psychology of Personal Constructs*, New York: Norton, 1955.

Kitto, C.M., and Boose, J.H., "An Implementation of Knowledge Acquisition Heuristics", *Boeing Computer Services Technical Report* BCS-G-2010-[36], 1986.

Kitto, C.M., and Boose, J.H., "Heuristics for Expertise Transfer: An Implementation of a Dialog Manager for Knowledge Acquisition", *Special Issue of the International*

Journal of Man-Machine Studies on the AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, to be published, 1987.

Klinker, G., Boyd, C., Genetet, S., and McDermott, J., "A KNACK for Knowledge Acquisition", *Carnegie-Mellon University Department of Computer Science Technical Report*, 1987.

Klinker, G., Bentolila, J., Genetet, S., Grimes, M., and McDermott, J., "KNACK - Report-Driven Knowledge Acquisition", *Special Issue of the International Journal of Man-Machine Studies on the AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,* to be published, 1987.

Marcus, S., "Taking Backtracking with a Grain of SALT", *Special Issue of the International Journal of Man-Machine Studies on the AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,* to be published, 1987.

Marcus, S. and McDermott, J., "SALT: A Knowledge Acquisition Tool for Propose-and-Revise Systems", *Carnegie-Mellon University Department of Computer Science Technical Report*, 1987.

Marcus, S., McDermott, J., and Wang, T. "Knowledge Acquisition for Constructive Systems", in *Proceedings of the Ninth Joint Conference on Artificial Intelligence*, Los Angeles, California, August, 1985.

Marcus, S., Stout, J., and McDermott, J.,"VT: An Expert Elevator Configurer that Uses Knowledge-Based Backtracking," *Carnegie-Mellon University Department of Computer Science Technical Report*, 1987 .

Musen, M.A., Fagan, L.M., Combs, D.M. and Shortliffe, E.H., "Using a Domain Model to Drive an Interactive Knowledge Building Tool", *Special Issue of the International Journal of Man-Machine Studies on the AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,* to be published, 1987.

Shaw, M.L.G. and Gaines, B.R., "Techniques for Knowledge Acquisition and Transfer", *Special Issue of the International Journal of Man-Machine Studies on the AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,* to be published, 1987.

Simpson, W.R. and Dowling, C.S., "WRAPLE: The Weighted Repair Assistance Program Learning Extension", *IEEE Design and Test*, pp 66-73, April 1986.

Stefik, M., Aikins, J., Balzer, R., Benoit, J., Birnbaum, L., Hayes-Roth, F., and Sacerdoti, E. "The Organization of Expert Systems. A Tutorial.", *Artificial Intelligence* 18(2), pp. 135-173, 1982.

Waterman, D., *A Guide to Expert Systems*, Reading, Mass: Addison-Wesley, 1986.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

Knowledge Acquisition for Evaluation Systems

Georg Klinker, Serge Genetet[1], and **John McDermott**[2]
Department of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

KNACK is a specialized knowledge acquisition tool that generates expert systems for evaluating different classes of designs. An important goal in the development of KNACK is that it acquires knowledge from domain experts without presupposing knowledge engineering skills on their part. During knowledge acquisition KNACK gains power by exploiting a presupposed problem solving method and a domain model. This paper describes KNACK's approach to automating the acquisition of a domain model as part of its knowledge acquisition strategy. To build a model of a domain, general understanding about evaluation is incorporated into KNACK. In an initial questioning session with domain experts KNACK customizes that knowledge and builds a preliminary model of the domain. Critical for KNACK's performance is its general understanding about evaluation and its ability to refine the preliminary domain model into a detailed structural and functional model of a particular domain. To get a better understanding of the means of evaluation and how to derive a domain model, KNACK was used to create a series of application systems in different domains. The experience gained with these tasks resulted in some data describing KNACK's performance and scope.[3]

1. Introduction

Existing expert systems have proven that AI techniques can be used to solve a variety of knowledge intensive problems. But expert systems are time-consuming to develop and difficult to maintain. A key issue in developing any expert system is how to update its large and growing knowledge base. It has been shown that a large knowledge base can be kept maintainable by

---

[1]Serge Genetet is now with Banque Internationale de Placement, 108 Boulevard Hausmann, 75008 Paris, France

[2]John McDermott is currently with Digital Equipment Corporation, 77 Reed Road, Hudson, Massachusettes, 01749

[3]This research was sponsored by the Defense Nuclear Agency (DNA) and the Harry Diamond Laboratories (HDL) under contract DNA001-85-C-0027, the Bares Foundation, Boeing Computer Services (BCS), and Digital Equipment Corporation (DEC). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DNA, HDL, the Bares Foundation, BCS, or DEC.

organizing it according to the different roles that knowledge plays [Chandrasekaran 83], [Clancey 83], [Neches 84]. Based on this realization a variety of knowledge acquisition tools have been produced during the past years to overcome those development and maintenance problems.

Existing knowledge acquisition tools focus on different aspects of the knowledge engineering task. KREME [Abrett 87] provides an environment for editing large knowledge bases. SEAR [van de Brug 86], AQUINAS [Boose 87], KRITON [Diederich 87], and TKAW [Kahn 87] integrate a variety of methodologies and tools for the development of expert systems into a workbench for a knowledge engineer.

Other knowledge acquisition tools try to automate the knowledge acquisition process. An automated knowledge acquisition tool typically interacts with domain experts directly. No knowledge engineer is necessary to translate the expert's knowledge into production rules. An automated knowledge acquisition tool further organizes the knowledge it acquires, and generates an expert system. The domain expert can also use it to test and maintain the program it generates. The critical feature of such a tool is that a domain expert can use it without having to know about programming in general and specific AI techniques. Examples of automated knowledge acquisition tools are TEIRESIAS [Davis 82], ETS [Boose 84], MORE [Kahn 85], MOLE [Eshelman 87], and SALT [Marcus 87]. These tools derive their power by presupposing the problem solving method of the expert systems they generate [McDermott 86], [Gruber 87]. Other automated knowledge acquisition tools like OPAL [Musen 87] and STUDENT [Gale] exploit an explicit domain model.

A useful distinction between the above knowledge acquisition tools is whether they help to create expert systems that either select or construct a solution [Clancey 84]. TEIRESIAS, ETS, MORE, MOLE, OPAL and STUDENT generate expert systems that select a solution from a given set of pre-enumerable candidates. SALT is an example of a knowledge acquisition tool for systems that construct a solution.

KNACK is a knowledge acquisition tool that can be used by domain experts to create expert systems that assist with the evaluation of different classes of designs. Like SALT, it can be used to develop expert systems that construct a solution compatible with a set of constraints. But whereas SALT generated expert systems produce designs from scratch, i.e., typically one designer has complete knowledge about all constraints a solution has to satisfy, KNACK generates evaluation systems. Evaluation systems assume that multiple designers are involved in a design task and each designer only knows a subset of the constraints a solution has to satisfy. The designers have to work together to construct a design. Evaluation systems are useful when designers have little or no knowledge about specialized design practices required by unfamiliar design techniques or environments. For example, this is the case when new techniques have been developed that are not yet common knowledge. Therefore, evaluation systems assist a designer in refining a given design to take into account aspects of an unfamiliar environment. They evaluate designs from a different point of view not anticipated in the original design. Evaluation systems assume that an incomplete design exists that may be suboptimal under a particular perspective, compare the parameters of a given design with constraints imposed by a given environment, and propose fixes in case some constraints are

violated.

Another difference between KNACK and the tools mentioned so far is KNACK's report-driven approach to acquiring knowledge. KNACK assumes that an expert can present his knowledge adequately in the form of a report. The expert must have a clear understanding of what constitutes an acceptable report describing and evaluating a design. This includes that the expert knows what information is needed, how to evaluate this information, and how a designer should present this information.

If categorized along the dimensions outlined above, KNACK exploits a presupposed problem solving method as well as an explicit domain model. Like TEIRESIAS, ETS, MORE, MOLE, SALT, and SEAR it presupposes and exploits the problem-solving methods and the knowledge roles of the expert systems it generates. A problem-solving method repesents the knowledge needed to control the task. It is defined, such, that the domain knowledge, organized in knowledge roles, does not include control knowledge. KNACK takes advantage of this separation of control knowledge and domain knowledge to free the expert from having to define the control knowledge and the implementation details. KNACK's presupposed problem-solving method for evaluation is described in [Klinker 87a].

Like OPAL and STUDENT, KNACK derives its power by exploiting an explicit domain model during knowledge acquisition. It differs from those systems in that no knowledge engineering expertise is required to build a model of a particular domain. This is achieved by incorporating a general understanding about evaluation in KNACK. In an initial questioning session with domain experts KNACK customizes that knowledge and builds a preliminary model of the domain. This is an interactive process and takes a few hours of the expert's time. During knowledge acquisition KNACK gains power by exploiting the domain model. To get a better understanding of the means of evaluation and how to derive a domain model, KNACK was used to create a series of evaluation systems for applications in different domains. The experience gained with these tasks shows that critical for KNACK's performance is its presupposed understanding of evaluation and its ability to refine the preliminary domain model into a detailed structural and functional model of a particular domain. This paper describes a unifying approach for a series of evaluation systems in different domains that were created using KNACK.

Section 2 introduces and compares the expert systems for different evaluation domains generated with KNACK. KNACK's report-driven approach for knowledge acquisition is described in section 3. Section 4 presents how KNACK uses its presupposed, general understanding of evaluation to acquire preliminary domain models, and section 5 describes KNACK's approach to refine a preliminary domain model into a detailed structural and functional model of a particular domain. Section 6 provides some data describing the complexity of the different evaluation domains and the expert's effort to generate the expert systems.

2. The WRINGERs

Each of the expert systems produced by KNACK is called a WRINGER. The purpose of a WRINGER is to assist a designer in refining a given, incomplete design to take into account aspects of an unfamiliar environment. It further presents this design, together with a preliminary design evaluation, in the form of a report. To evaluate designs a WRINGER must have available an initial description of the design to be evaluated. Thus, a WRINGER first gathers the information describing an existing design and then evaluates the information. If as a result of the evaluation, additional information is required, a WRINGER gathers that information. After that, another evaluation is performed. This iterative process ends when the designer is satisfied with the design.

To gather the information describing a specific design, a WRINGER uses strategies to elicit information from the designer or to infer it. For example, it asks questions or computes numeric values using formulas. As it progresses, the gathering of information is driven by previously elicited information. This is a data-driven approach that modifies a WRINGER's behavior according to the information specific to each design it is applied to.

The collected information is evaluated by the WRINGER for validity, consistency, completeness, and possible design flaws, i.e., a WRINGER checks the information describing a design for violations of constraints imposed by a given environment. If indications of design flaws are found, a WRINGER points them out to the designer together with suggestions for improving the design. If the designer agrees with the fix, a WRINGER updates the design description using a dependency network. Finally, when the designer is satisfied with the design, a WRINGER generates a report describing and evaluating the design.

KNACK has been or is being used by knowledge engineers to generate the following WRINGERs:

The Data Item Description WRINGER Family:[4]WRINGERs have been developed for different stages in the design of electromechanical systems for the nuclear hardening domain. Nuclear hardening involves the use of specific engineering design practices to increase the resistance of an electromechanical system to the environmental effects generated by a nuclear weapon. Designers of electromechanical systems usually have little or no knowledge about the specialized analytical methods and engineering practices of the hardening domain. The purpose of the WRINGERs is to assist a designer in improving given designs of electromechanical systems that may be suboptimal from a hardening perspective. The WRINGERs assume that the initial design describes a technically functional system. They evaluate the design from a hardening perspective. The suggested improvements are either extensions to the design or recommendations for using different design components. The WRINGERs present the design, together with the results of the evaluation, in the form of a technical document that meets government requirements.

---

[4]The WRINGERs of the Data Item Description WRINGER Family were created with a previous implementation of KNACK, reported in [Klinker 87a], and are now being tested by the organization that will use them. The domain experts involved in that task have some programming experience but no knowledge about AI techniques. The experts use KNACK to modify and update the expert systems.

The first WRINGER, a PROGRAM PLAN writer, evaluates and presents the primary top level report covering all phases of the design project. It took 7 person-days to create the WRINGER with KNACK. Its knowledge base contains 795 OPS5 [Forgy 81] rules. The second expert system, a DESIGN PARAMETERS REPORT writer, evaluates and presents a detailed description of an electromechanical system ranging from the level of major components to the level of individual semiconductors. It took 21 person-days to create the WRINGER with KNACK. Its knowledge base contains 1446 OPS5 rules. The last WRINGER, a TEST PLAN writer, produces a plan to confirm the hardness of a design. This includes a list of the design components to be tested, a description of the tests to be performed, and the expected test results. It took 8 person-days to create the WRINGER with KNACK. Its knowledge base consists of 230 rules.

The XNET-Design WRINGER: A WRINGER is being developed to assist a sales-person with the design and configuration of computer networks. In general, a sales-person has a good understanding about aspects like costs, compatibility, and extensibility when he is designing a network which suits his customer best. But usually he has little or no knowledge about the technical aspects involved. The purpose of the WRINGER is to assist a sales-person in improving his design of computer networks that may be suboptimal from a technician's perspective. The WRINGER assumes that the initial design describes the computing environment but might not be a technically functional computer network. It evaluates the design description from a technician's perspective. The suggested improvements are either extensions to the design or recommendations for different interconnections between design components. The WRINGER's output is a list of generic network components and their interconnections serving as input for a program that will select the specific parts. The WRINGER is in the very first stage of development. Data that describe KNACK's performance are not yet available.

The Software Requirements WRINGER: A WRINGER is being developed to assist a systems analyst with the definition of requirements for software. Defining requirements for new software is a very complex process. It involves functionally decomposing the software into basic modules, defining the data requirements, and integrating the new software with the existing software environment. One systems analyst alone might not have enough knowledge about the existing software environment. The purpose of the WRINGER is to assist a systems analyst in refining the requirements for software systems that may be suboptimal given the existing software environment. The WRINGER assumes that the initial design describes the new software on a high level of abstraction. It supports the systems analyst in functionally decomposing that description into basic modules and defining the data requirements for the modules. The WRINGER evaluates the design as to whether it is compatible with the existing software environment. The suggested improvements are refinements to the requirements of the new software. The WRINGER produces a technical document describing the requirements for the software system. The document further contains an executive summary with an opinion about whether the new software will be a valuable enhancement of the existing software. It took 18 person-days to create the WRINGER with KNACK. Its knowledge base consists of 291 rules.

The Project Progress Report WRINGER: A WRINGER is being developed to assist a project leader with the assessment of a project's progress. A project leader might not have enough

experience to create a project plan and assess the progress of a project. Also, he might not have enough knowledge to integrate his project into the broader objective of his management. The purpose of the WRINGER is to assist a project leader in refining a project plan that may be suboptimal from a management perspective. The WRINGER assumes that an initial proposal can be provided. It supports the project leader in creating and updating the project plan according to the progress of the project. The WRINGER evaluates the plan from a management perspective. The suggested improvements are refinements or changes to the plan. The WRINGER produces a project plan and periodical progress reports that will allow management to assess the progress of a project. It took 19 person-days to create the WRINGER with KNACK. Its knowledge base consists of 429 rules.

The Business Plan WRINGER: A WRINGER is being developed to assist an entrepreneur in the preparation of business plans. The first step in creating a business is to seek investment capital. For this purpose, entrepreneurs generate business plans. A business plan contains information on the planned business, e.g. the industry, the product, the market and marketing plan, production, personnel, and financial projections. An entrepreneur usually has little knowledge about how to create a business plan. The purpose of the WRINGER is to support an entrepreneur in developing a business plan to secure investment capital from venture capitalists. The WRINGER assumes that the entrepreneur can provide a description of the product and the goals of the proposed business. It will then support the entrepreneur in creating the business plan. The WRINGER evaluates the plan from a venture capitalist's perspective. It produces a document containing the necessary details and justifications to demonstrate the proposed business to an investor. The information is presented in a way appealing to an investor. The WRINGER is in the very first stage of development. Data that describe KNACK's performance are not yet available.

3. KNACK

KNACK is a knowledge acquisition tool that can be used by domain experts to create expert systems that assist with the evaluation of different classes of designs. It combines and uses existing AI techniques to derive a general description how to evaluate designs from a specific sample description. This is a process of abstraction (e.g. the specific sample description is variabilized) and completion (e.g. signs of incompleteness lead to the elicitation of additional knowledge). The following is a summary of the approach KNACK takes to acquire knowledge. A more detailed description can be found in [Klinker 87b].

An important goal in the development of KNACK is, that it acquires knowledge from domain experts without presupposing knowledge engineering skills on their part. This is achieved by incorporating general understanding about evaluation into KNACK. The incorporated knowledge describes the evaluation task on a high level of abstraction. It views evaluation as partly analytic and partly constructive, i.e., the goal of the evaluation task is to determine whether a design will function in a given environment and how the design can be improved so that it will function in that environment.

In an initial interaction, KNACK uses its general understanding of evaluation to conduct an interview with the expert. That interview results in a preliminary model of a particular evaluation

domain. The domain model describes the concepts and the vocabulary the expert uses in performing an evaluation task. KNACK also requires a sample report as an initial input. The sample report is a document that exemplifies the description and the evaluation of a particular design. It contains a description of the design and the given environment, a detailed evaluation of the design with regard to the environment, and suggestions to improve the design in case design flaws are found. This report is typed in to a text editor by any person familiar with keyboard text entry and KNACK divides the report into fragments corresponding to paragraphs.

Once the sample report is typed in and an initial domain model is defined, KNACK interacts with a domain expert to integrate the sample report with the domain model on a fragment by fragment basis. This involves extracting the report's basic structure and integrating the domain model with the report fragments. The technique uses simple heuristics to match concepts in the domain model with instances in fragments of the sample report. The analytical power provided by the heuristics suffices to acquire knowledge without turning to a sophisticated natural language interface. The integration process generalizes the sample report, making it applicable to different systems.

To demonstrate its understanding of the sample report and to predict and exemplify the performance an expert can expect from the WRINGER he is creating, KNACK instantiates the generalized report with known concept representatives taken from the domain model. It displays several differently instantiated examples for each generalized report fragment. The expert edits any examples that make implausible statements. KNACK uses this feedback as additional knowledge to correct its generalizations and refine the domain model.

Once the expert accepts KNACK's understanding of the sample report, KNACK elicits knowledge about how to customize the generalized sample report for a particular application. The expert defines strategies that a KNACK generated expert system, a WRINGER, will use to acquire values instantiating the concepts in the generalized fragments. Experts define strategies in the same way that report fragments are defined: by typing in samples. Each strategy describes a way to determine a representative of a concept detected in generalized fragments and includes instructions about valid possible values. Among others, a strategy can acquire concept representatives by asking questions, interpreting a graphical design description, asking the designer to fill in the slots of a table or form, or asking the user to choose from the items in a menu. It can infer concept representatives by directly applying specific domain knowledge, computing numeric values using formulas, or referring to a database. KNACK generalizes the strategies and displays some example instantiations of them for review and correction by the expert.

KNACK's knowledge acquisition approach results in a knowledge base the generated WRINGER expert system can use to evaluate a range of designs and to present the results in the form of a report. However, the sample report covers only one simple design and almost certainly lacks some important concepts needed for the evaluation of a broader range of designs. For this reason, KNACK searches the knowledge base for report fragments or strategies that indicate gaps or conflicts with its domain model. Since an apparent gap found during the process might be filled in later in the process, this review of the knowledge base is most relevant at the end of the acquisition process. When a conflict was detected or an

indication of a gap was found, KNACK asks the expert to correct either the fragment, the strategy, or the domain model. In cases where the domain model is changed, KNACK reviews all fragments or strategies that use the changed concept or relation to propagate the change through the knowledge base automatically, making guesses when ambiguities arise. On the other hand, when the expert adds or changes report fragments or strategies, KNACK processes them through the integration of the domain model, display of examples, strategy definition, and checking.

The above description of KNACK's approach for knowledge acquisition reveals that the generalization process is critical for KNACK's performance. The technique uses simple heuristics to replace fixed text components of the sample report with variables denoting concepts and concept characteristics. The heuristics take into account the structure of the sample report and previous generalizations. They look for keywords, known concepts, and concept representatives in the sample report. Since the concepts and concept representatives are described in the domain model, the domain model is the heart of the generalization process. A successful generalization requires that the concepts and concept representatives in the domain model closely correspond to the terms used in the sample report. Moreover, the domain model must contain a detailed structural and functional description of the evaluation task. The structural part of the domain model describes a taxonomy of the vocabulary and terms experts use. This includes the relevant parameters of a design and the environment, the constraints, and possible fixes for violated constraints. The functional part of the domain model describes procedures how to determine, compare and propagate relevant parameters.

Since, an important goal in our research is to make the use of KNACK independent from knowledge engineering skills, KNACK does not assume that the domain expert can define a complete and detailed domain model consistent with the sample report. It expects that the expert can provide an initial vocabulary that contains some of the terms used in the evaluation task. KNACK gradually augments that preliminary model during knowledge acquisition into a domain model describing a design and the evaluation process. The remaining part of the paper describes KNACK's approach to acquire a preliminary domain model and to gradually refine it into a detailed structural and functional model of the domain.

4. Acquiring a Domain Model

To acquire the domain model, KNACK conducts an interview with the expert. The interview is driven by KNACK's general understanding of the evaluation task. That understanding describes the knowledge common to a range of evaluation domains. The example in Figure 4-1 gives a flavour of the knowledge applicable to a range of tasks: A design has to be refined to take into account aspects of an unfamiliar environment. It comprises a set of interrelated design components. Design components have design properties. An environment comprises environment components. They are further described by environment properties. Design and environment properties define the evaluation criteria and are compared to some other properties. This defines the constraints a design has to satisfy. Properties can comprise property components. Design components can propose a fix for properties or property components. A fix changes a design or suggests a design extension and, thus, modifies design or environment properties.

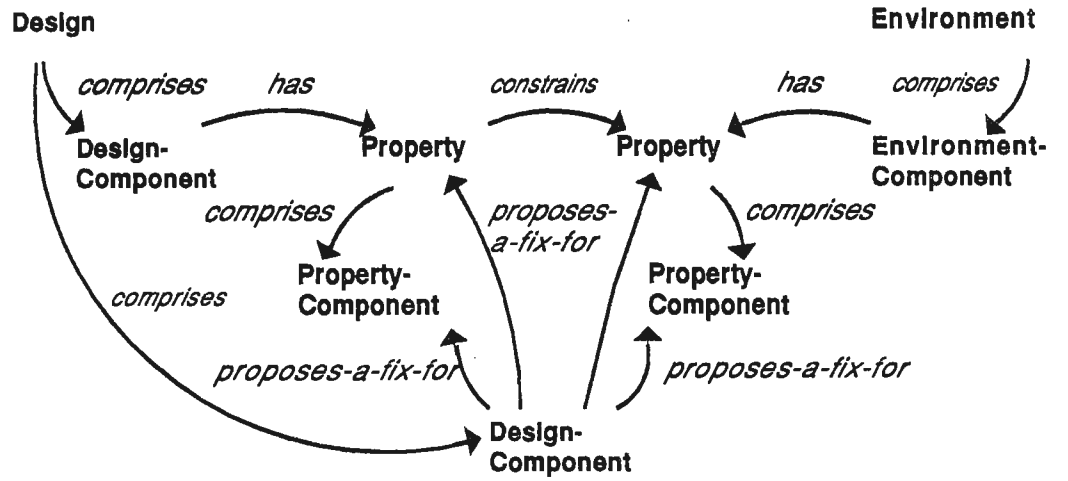


Figure 4-1: KNACK's Presupposed Understanding of Evaluation

KNACK uses the above described general understanding in the form of generic questions to acquire the vocabulary experts are familiar with. The expert's answers customize KNACK's abstract knowledge into a preliminary model of a particular domain. The preliminary model describes concepts, concept characteristics, concept representatives, and constraints. A typical question would be: "How would you like to refer to different design components?" That question introduces new concepts. Concepts denote design components, environment components, concept properties, and fixes in case constraints are violated. They establish the vocabulary experts use in performing their evaluation task. Figure 4-2 presents a part of the preliminary domain model for the hardening domain. That example contains the concepts subsystem, interface-circuit, cable, etc. These concepts describe design components. Coupling denotes an environment component. Enclosure, optical-coating, honeycomb, etc. are examples for the terms experts use for fixes. Concept properties are: safe-energy, threat-energy, safe-voltage, and threat voltage. Other questions are used to define constraints between design components and environment components. Constraints represent the criteria to be satisfied by a design. The example in figure 4-2 states that the safe-energy property constrains the threat-energy property and the safe-voltage property constrains the threat-voltage property. Questions are also used to determine concept representatives. Concept representatives are examples for concepts. For example: computer, modem, radio, and motor generator are representatives for subsystems. KNACK's generalization heuristics look for those examples in the sample report. Further questions acquire concept characteristices that describe concepts further and can be used to determine design and environment properties. In the example of figure 4-2, an enclosure is described by material, thickness, relative-conductivity, and minimum-thickness.

Figure 4-3 presents a part of the preliminary domain model for the software requirements domain. The design of the new software is described in terms of the functions it will perform, their input data, and their output data. These terms are also used to denote fixes. The organizations of a company and their activities determine the environment. Constraints ensure

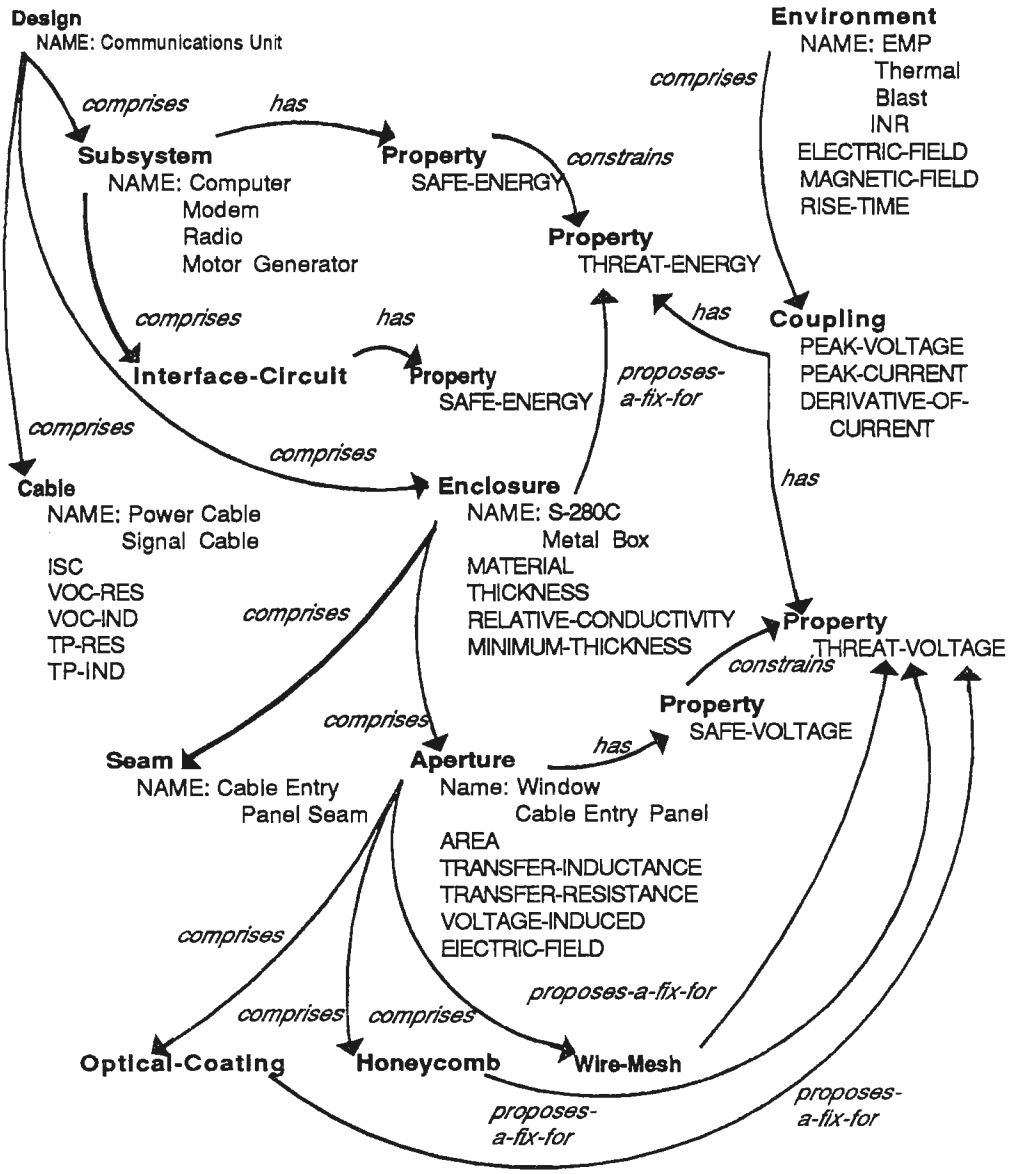


Figure 4-2: Model for the Hardening Domain

that input data for the new software are either produced by existing organizations or other functions of the planned software. "Do system design" and "Parts Library" are examples of concept representatives for functions or input data. Concept characteristics like summary, frequency, volume, format, and security define the detailed requirements for input data.

5. Refining the Domain Model

The initial domain model acquired through the interview is not sufficient for a successful generalization process. It represents a structural description of the task i.e., the vocabulary and the terms experts use, but does not contain any functional knowledge, i.e., how experts obtain and propagate design and environment parameters. It is further likely that the initial structural

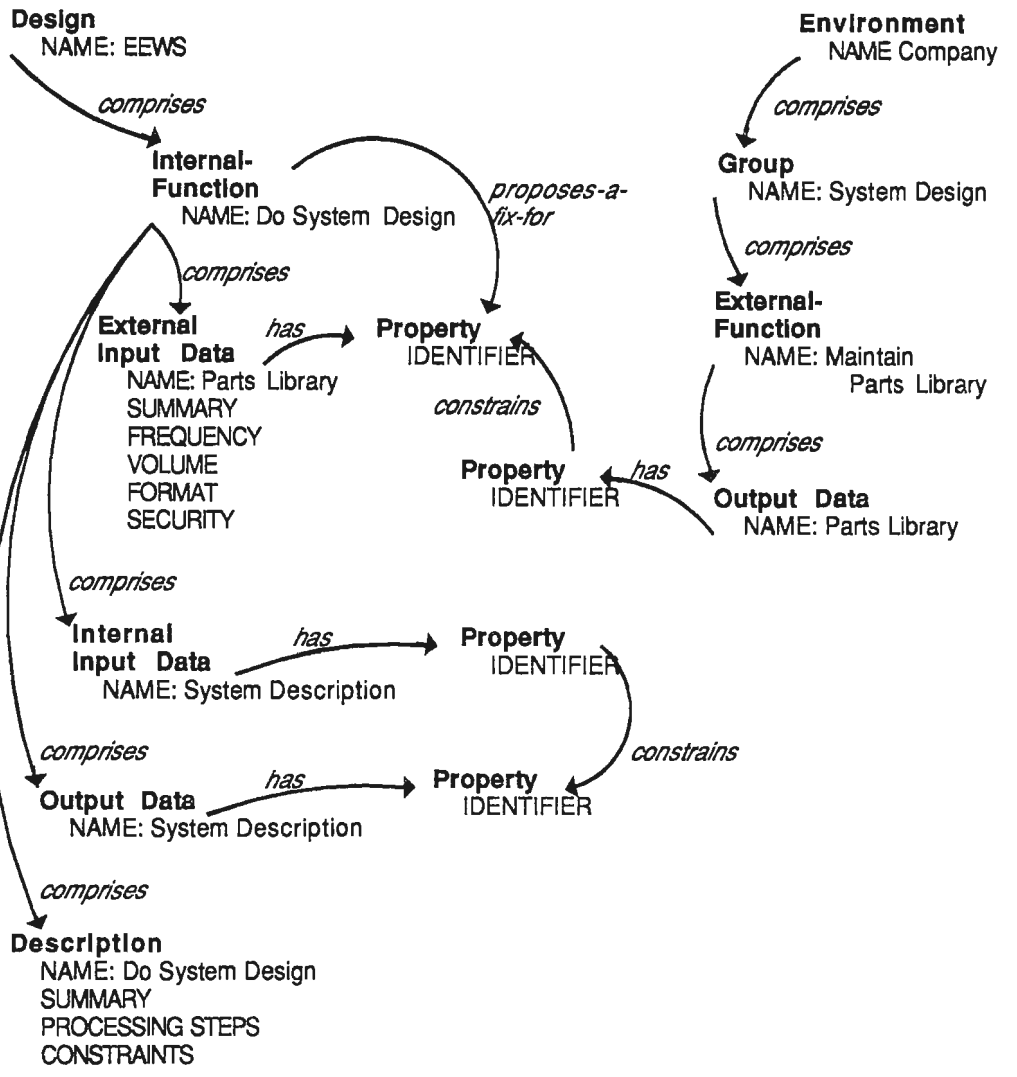


Figure 4-3: Model for the Software Requirements Domain

model is incomplete and not detailed enough. During knowledge acquisition KNACK augments the initial domain model to include the functional aspects of the evaluation domain and to obtain a more detailed structural model. It gradually specializes the domain model to represent the expert's understanding of how a particular environment interacts with different design components.

KNACK asks the expert to define strategies a WRINGER can use to acquire instantiation values that customize the concepts in the generalized report for a particular application. The strategies define how to obtain, propagate, and compare design and environment parameters. For example, questions determine the relevant design components or a formula computes some design property given some design characteristics. Once generalized, strategies provide the functional knowledge about the domain.

KNACK predicts and exemplifies the performance an expert can expect from the WRINGER he is working to create. It instantiates the concepts of the generalized fragments with known concept representatives taken from the domain model and displays several differently instantiated examples of each generalized report fragment or strategy. The expert edits any examples that make implausible statements. KNACK treats such events as incorrect use of the knowledge base and interprets the corrections as new knowledge to update the generalization and improve the domain model. For example, KNACK may create new concepts or concept representatives when a correction introduces a variability it cannot understand, given the current domain model. Extensions can be new concepts, new characteristics for known concepts, and restrictions on existing relations between representatives of two concepts. The model serves as a collection of examples suggesting guesses for KNACK as to the form of the extension. Or, if the expert's correction indicates that values from the domain model combine too loosely, KNACK adds a constraint to the model, restricting possible combinations. A correction also can imply that KNACK's guess about the identity of a concept is wrong, leading to its retraction and the introduction of a new, initially less probable guess. Applying the new knowledge, the generalization is instantiated again and redisplay of several examples gives the expert immediate feedback on the effects of the knowledge base modification. KNACK uses the correction process to complete its model of the domain.

Figures 5-1 and 5-2 show parts of the refined domain models for the hardening domain and software requirements domain.

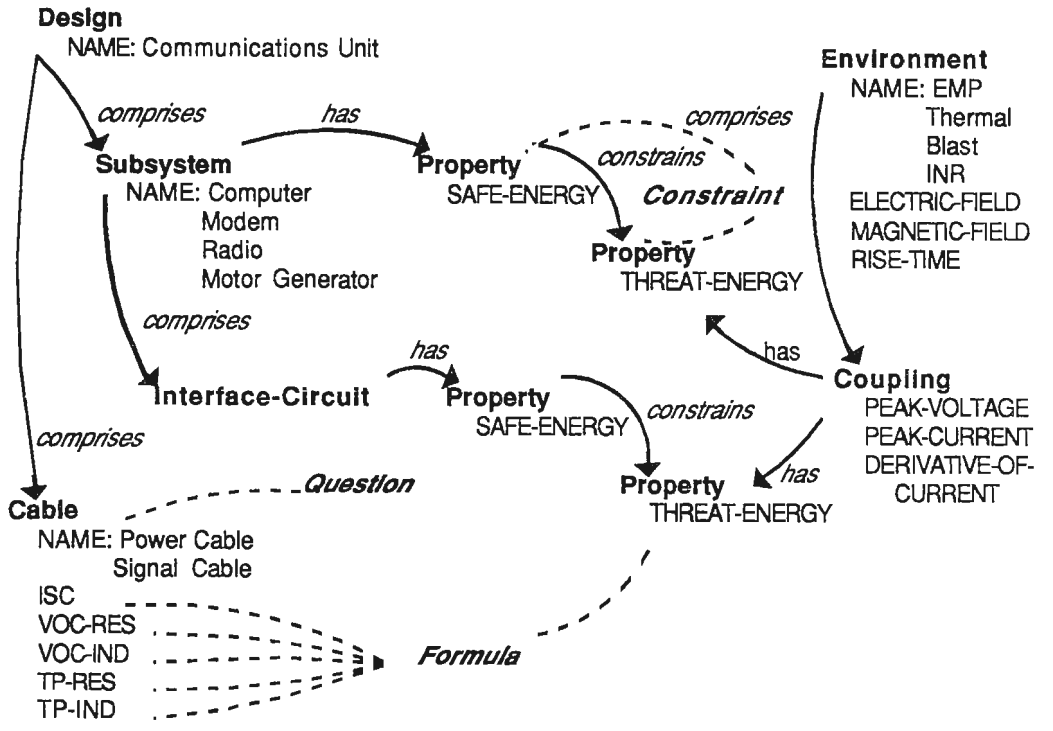


Figure 5-1: Refined Model for the Hardening Domain

The dotted lines indicate the functional information added to the domain model through the definition of strategies. For example, in figure 5-1, a question determines what kind of cables are used in a particular design, or a formula uses the values of the cable characteristics to compute the threat-energy property of the coupling environment component.

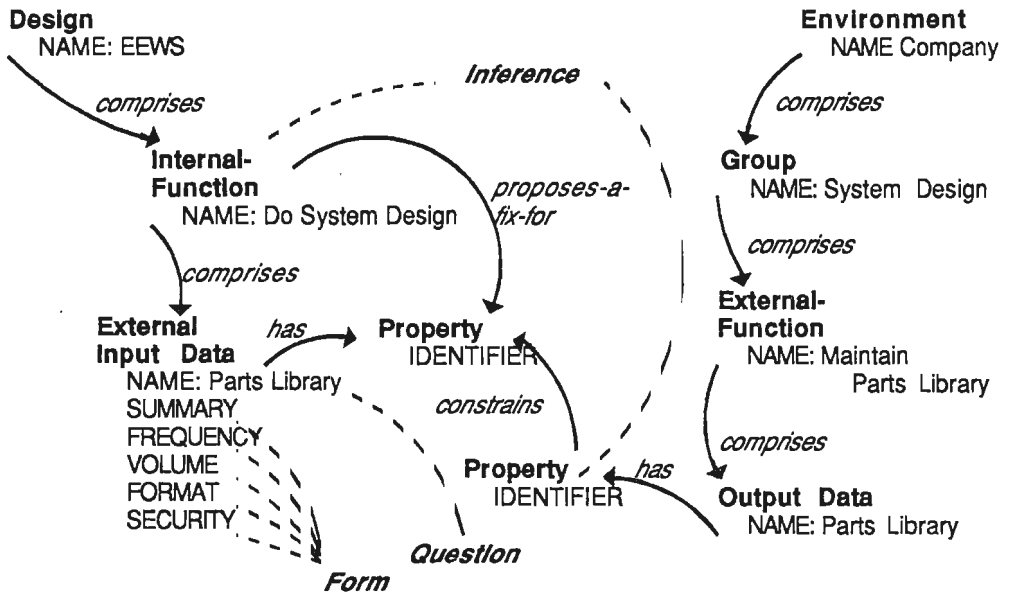


Figure 5-2: Refined Model for the Software Requirements Domain

The examples in figure 5-2 indicate that external input data are determined by a question. Their characteristics can be obtained by filling in the slots of a form. The figure also contains an example of an inference: whenever the planned software requires input data that are not available, a function producing the necessary data will be suggested as an extension to the new software. The systems analyst decides whether to include that function into the software requirements.

6. Some Performance Data

This section gives some impression of KNACK's performance in creating the Program Plan WRINGER (PP WRINGER), Design Parameters Report WRINGER (DPR WRINGER), Test Plan WRINGER (TP WRINGER), Software Requirements WRINGER (SR WRINGER), and Project Progress Report WRINGER (PPR WRINGER). The Business Plan WRINGER and XNET-Design WRINGER are in the very first stages of development. Data that describe KNACK's performance in creating them are not yet available. The data of Table 6-1 describe the complexity of the domains, Table 6-2 contains some data about the complexity of the generated knowledge bases, and Table 6-3 summarizes the effort involved in creating the expert systems with KNACK.

Table 6-1 gives some impression of the complexity of the domains for the five WRINGERs. It describes the input the experts had to provide in terms of the sample report, the domain model,

and the sample strategies. Strategies can be interactive, i.e., they elicit information from the WRINGER users, or autonomous, i.e., they infer information based on previously provided information. Examples of interactive strategies are: questions, graphical design descriptions, menus, forms, or tables. Examples of autonomous strategies are: inferences, database lookups, or formulas.

	PP WRINGER	DPR WRINGER	TP WRINGER	SR WRINGER	PPR WRINGER
Number of fragments in the sample report	237	455	88	203	113
Average size of each fragment in words	9.5	14.7	14.3	10.2	8.1
Number of concepts	43	92	28	55	109
Average number of characteristics for each concept	2.3	3.7	2.5	3.6	1.4
Number of interactive strategies	72	152	35	21	92
Number of autonomous strategies	22	159	4	32	41

Table 6-1: Complexity of the Domain

Table 6-2 describes the generated knowledge base for the five WRINGERs. The size of the knowledge base is determined by the number of OPS5 rules it contains. The conditionality of a rule is described by the number of its condition elements. Each condition element can be instantiated by a concept. The complexity of a condition element is defined by the number of characteristics which describe a concept. The action part of a rule is described by the number of actions the rule performs. An action either creates a new concept or modifies an existing concept.

	PP WRINGER	DPR WRINGER	TP WRINGER	SR WRINGER	PPR WRINGER
Number of rules	795	1446	230	291	429
Average number of condition elements per rule	3.4	3.8	1.1	1.9	2.9
Number of characteristics per condition element	2.0	2.3	1.5	2.0	2.5
Number of actions per rule	7.2	3.0	2.9	4.1	1.9

Table 6-2: The Knowledge Base

Finally, Table 6-3 gives some impression of the time involved to generate a WRINGER using KNACK. Creating WRINGERs is an iterative process. Whenever a WRINGER reveals inadequacies, KNACK is used to improve it. Table 6-3 shows the time spent to generate the

WRINGERs described in section 2 and in the tables 6-1 and 6-2. This includes the effort for the initial input (sample report, domain model, and sample strategies), the generalization process (sample report and sample strategies), and the review of the knowledge base. The effort for the generalizations includes the expert's corrections to the sample instantiations of the generalized fragments and strategies. Since the PP WRINGER, DPR WRINGER, and the TP WRINGER were created with a previous implementation of KNACK, no detailed data are available.

	PP WRINGER	DPR WRINGER	TP WRINGER	SR WRINGER	PPR WRINGER
Number of days to create the sample report	--	--	--	2	3
Number of days to create the preliminary domain model	--	--	--	4	5
Number of days to create the sample strategies	--	--	--	3	1
Number of days to generalize the sample report	--	--	--	4	5
Number of days to generalize the sample strategies	--	--	--	4	5
Number of days to review the knowledge base	--	--	--	1	--
Total	7	21	8	18	19

Table 6-3: Effort

7. Conclusion

This paper described KNACK's approach to automate the acquisition of a domain model as part of its knowledge acquisition strategy. KNACK has been or is being used to create applications systems for several different domains. Each application improved our understanding of evaluation and had some implications on the development of the KNACK tool. An important goal in our research is to make the use of KNACK independent of knowledge engineering expertise. Since the heart of KNACK's generalization process is a detailed structural and functional model of the domain, we will continue to make KNACK less sensitive to incomplete domain models. This includes getting a better understanding of the knowledge common to a range of evaluation tasks and improving the heuristics used in the generalization process. We will further extend KNACK's ability to use the expert's corrections as well as the generalized strategies to refine the preliminary model into a structural and functional representation of a particular evaluation domain.

The different applications of KNACK (described in section 2) gave us some insight into KNACK's scope. At a first glance, those applications seem to be quite different. But a closer look reveals that they all meet some common requirements. The task is constructive evaluation, i.e., an existing plan or design has to be evaluated to determine whether it meets additional constraints not antcipated in the original design. The evaluation is constructive

because fixes can be suggested in case of constraint violations. The use of KNACK imposes some requirements on the expert: The expert must be able to provide an initial model of the domain, he must be able to express some of his knowledge in the form of a sample report, and he must be able to define strategies that a WRINGER can use to instantiate concepts with values describing a particular plan or design.

Acknowledgments

We would be remiss if we did not mention our co-workers in this project: Casey Boyd, David Dong, Don Kosy, Jon Maiman, and Bob Schnelbach are members of the group and made significant contributions. We would like to thank Larry Eshelman who reviewed an earlier draft of this paper.

References

[Abrett 87] Abrett, G., and M. Burstein.
The KREME Knowledge Editing Environment.
International Journal of Man-Machine Studies (), to appear 1987.

[Boose 84] Boose, J.
Personal construct theory and the transfer of human expertise.
In *Proceedings of the National Conference on Artificial Intelligence*. Austin, Texas, 1984.

[Boose 87] Boose, J., and J. Bradshaw.
Expertise Transfer and Complex Problems: Using AQUINAS as a Knowledge Acquisition Workbench for Expert Systems.
International Journal of Man-Machine Studies 26(1):3 - 28, 1987.

[Chandrasekaran 83]
Chandrasekaran, B.
Towards a taxonomy of problem solving types.
AI Magazine 4(1):9-17, 1983.

[Clancey 83] Clancey, W.
The advantages of abstract control knowledge in expert system design.
In *Proceedings of the 3rd National Conference on Artificial Intelligence*. Washington, D.C., 1983.

[Clancey 84] Clancey, W.
Classification problem solving.
In *Proceedings of the 4th National Conference on Artificial Intelligence*. Austin, Texas, 1984.

[Davis 82] Davis, R. and D. Lenat.
Knowledge-Based Systems in Artificial Intelligence.
McGraw-Hill, 1982.

[Diederich 87] Diederich, J., I. Ruhmann, and M. May.
KRITON: A Knowledge Acquisition Tool for Expert Systems.
International Journal of Man-Machine Studies 26(1):29 - 40, 1987.

[Eshelman 87] Eshelman, L., D. Ehret, J. McDermott, and M. Tan.
MOLE: A Tenacious Knowledge Acquisition Tool.
International Journal of Man-Machine Studies 26(1):41 - 54, 1987.

[Forgy 81] Forgy, C.L.
OPS5 user's manual.
Technical Report, Carnegie-Mellon University, Department of Computer Science, 1981.

[Gale] Gale, W.
Knowledge Based Knowledge Acquisition for a Statistical Consulting System.
International Journal of Man-Machine Studies 26(1):55 - 64, .

[Gruber 87] Gruber, T., and P. Cohen.
Design for Acquisition: Principles of Knowledge System Design to Facilitate Knowledge Acquisition.
International Journal of Man-Machine Studies 26(2):143 - 159, 1987.

[Kahn 85] Kahn, G., S. Nowlan and J. McDermott.
MORE: an intelligent knowledge acquisition tool.
In *Proceedings of Ninth International Conference on Artificial Intelligence.* Los Angeles, California, 1985.

[Kahn 87] Kahn, G., E. Breaux, R. Joseph, and P. DeKlerk.
An Intelligent Mixed-Initiative Workbench for Knowledge Acquisition.
International Journal of Man-Machine Studies (), to appear 1987.

[Klinker 87a] Klinker, G., J. Bentolila, S. Genetet, M. Grimes, and J. McDermott.
KNACK - Report-Driven Knowledge Acquisition.
International Journal of Man-Machine Studies 26(1):65 - 79, 1987.

[Klinker 87b] Klinker, G., C. Boyd, S. Genetet, and J. McDermott.
A KNACK for Knowledge Acquisition.
In *Proceedings of Sixth National Conference on Artificial Intelligence.* Seattle, Washington, 1987.

[Marcus 87] Marcus, M.
Taking Backtracking with a Grain of SALT.
International Journal of Man-Machine Studies (), to appear 1987.

[McDermott 86] McDermott, J.
Making expert systems explicit.
In *Proceedings of 10th Congress of the International Federation of Information Processing Societies.* Dublin, Ireland, 1986.

[Musen 87] Musen, M., L. Fagan, D. Combs and E. Shortliffe.
Using a Domain Model to Drive an Interactive Knowledge Editing Tool.
International Journal of Man-Machine Studies 26(1):105 - 121, 1987.

[Neches 84] Neches, R., W. Swartout, and J. Moore.
Enhanced maintenance and explanation of expert systems through explicit models of their development.
In *Proceedings of IEEE Workshop on Principles of Knowledge-based Systems.* Denver, Colorado, 1984.

[van de Brug 86] van de Brug, A., J. Bachant, J. McDermott.
The Taming of R1.
IEEE Expert 1(3), 1986.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

Knowledge Acquisition as Knowledge Assimilation

Lawrence S. Lefkowitz
Victor R. Lesser
Computer and Information Science Department
University of Massachusetts
Amherst, MA 01003
larry@umass-cs.edu
lesser@umass-cs.edu

Abstract

The assimilation of information obtained from domain experts into an existing knowledge base is an important facet of the knowledge acquisition process. Knowledge assimilation requires an understanding of how the new information corresponds to that already contained in the knowledge base and how this existing information must be modified so as to reflect the expert's view of the domain. This paper describes a system, K^n_{Ac}, that modifies an existing knowledge base through a discourse with a domain expert. Using heuristic knowledge about the knowledge acquisition process, K^n_{Ac} anticipates modifications to existing entity descriptions. These anticipated modifications, or *expectations*, provide a context in which to assimilate new domain information.

1 INTRODUCTION

An often overlooked aspect of the knowledge acquisition process is the assimilation of information presented by the domain expert into an existing knowledge base. Consider the typical means by which knowledge bases are currently constructed. This usually involves a series of dialogs between an expert, or experts, in the application domain and a knowledge engineer familiar with the target expert system. The knowledge engineer's task is the modification of the expert system's knowledge base so as to reflect the domain expert's knowledge. To a large extent, this knowledge acquisition task may be viewed as a recognition problem. All of the problems facing other recognition systems are

This research was supported, in part, by the External Research Program of Digital Equipment Corporation, by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and by the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under Contract No. F 30602-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC).

present here as well, including: noisy data (i.e., incomplete or inaccurate information), ambiguous interpretations, and the need to produce intermediate results before all the data is available. Thus, a significant portion of this interactive knowledge acquisition task is a matching problem: How does the expert's description of the domain correlate with the description contained in the knowledge base? How should the knowledge base be modified based on new information from the expert? What should be done when the expert's description differs from the existing one?

K^n_{Ac} is a system that we have developed that implements this knowledge assimilation approach to knowledge acquisition. It was developed to assist in the construction of knowledge bases for the POISE [Croft *et al.*, 1982] intelligent interface system. These knowledge bases use a frame-like representation, described more fully in [Croft and Lefkowitz, 1984,Lefkowitz, 1987] to describe *tasks*, *objects* and *relationships* in the application domain. POISE's initial knowledge bases, for the office automation and software engineering domains, were created by hand from interviews between a knowledge engineer and the appropriate domain experts. Transcriptions of these interviews were examined and the results served as the basis of the K^n_{Ac} system.

It is important to note that the goal of the domain expert was *not* to modify POISE's knowledge base; this was the knowledge engineer's role. The expert simply presented the domain information, e.g., descriptions of tasks, objects, etc., and responded to questions and comments from the knowledge engineer. The burden of assimilating the information, that is, recognizing where it fit into the existing knowledge base and what additions or modifications were needed, was not placed upon the domain expert. (Contrast this to approaches such as [Eshelman *et al.*, 1986,Ginsberg *et al.*, 1985,Kahn *et al.*, 1985].) By modeling the knowledge engineer's role in this task, K^n_{Ac} attempts to provide this same support.

Consider the opening portion of a discourse in which the expert, the principal clerk of an academic department, is describing the procedure for being reimbursed for business-related travel expenses.

> "O.K. — on travel. The proper way of doing it, if it's out of state, is that a travel authorization should be issued before the trip."

From this information one can conclude that some unnamed task consists of two temporally ordered steps. However, it is not clear what modifications need be made to the knowledge base to reflect this information.

If the knowledge base is examined (prior to this interview), a description of this reimbursement process will be found (see Figure 1). In this simplified view of the task, which knows nothing about a "travel authorization", the traveler simply goes on a trip and gets reimbursed. Though the knowledge engineer may realize that the clerk and this description are describing the same task, it is not readily apparent from the two descriptions. Matching such descriptions, and recognizing the implied modifications,

EVENT TAKE-A-TRIP-AND-GET-PAID
STEPS: (TAKE-A-TRIP GET-REIMBURSED)
TEMPORAL-RELATIONSHIPS:
((TAKE-A-TRIP *before* GET-REIMBURSED))
CONSTRAINTS: (···)
ATTRIBUTES: ((TRAVELER ···) (COST ···) (DESTINATION ···))

Figure 1: Knowledge Base Event Description

are central to the assimilation process.

To accomplish this, K^n_{AC} was required to perform two basic tasks: 1) recognizing where the expert's information fits into the existing knowledge base, and 2) appropriately modifying the existing knowledge so that it reflects the expert's view of the domain. Determining where the expert's information fits into the existing knowledge requires that the new information be matched against the existing information. To avoid matching the new information against the entire (existing) knowledge base, the most likely candidate matches must be selected. Furthermore, since the goal of a knowledge acquisition discourse is the modification of the knowledge base, exact matches between the new and the existing information are not always expected.

Thus, the procedure for matching the expert's entity descriptions with those already in the knowledge base must be specialized for knowledge acquisition. K^n_{AC}'s matching and match evaluation procedures are described in Section 3. Discrepancies between these descriptions may imply needed modifications and need not degrade a match, especially if the discrepancies (or the implied modifications) can be predicted. Anticipated modifications, or *expectations*, arise from an understanding of the knowledge acquisition process. They can be derived from the state of the existing knowledge base, from cues in the discourse, from previous modifications to entity descriptions, or from the state of the knowledge acquisition task. The generation and management of these expectations is described in Section 4. Finally, the status of this work and its contributions to the knowledge acquisition task are summarized in Section 5.

2 THE K^n_{AC} SYSTEM

In this section, the basic architecture and functionality of the K^n_{AC} system[1] is presented. During each cycle of the K^n_{AC} system, descriptions of domain entities are

[1] Figure 2 contains the architecture of the K^n_{AC} system. The parenthesized numbers in this paragraph (e.g., (1)) refer to this figure.

accepted from the user (1) and compared with entities in the existing knowledge base (2). (Figure 4 contains a portion of this knowledge base.) These candidate entities (3) are selected based on K^n_{Ac}'s expectations of changes to the knowledge base. The comparisons (4) are evaluated both in terms of how well they match and the extent to which the differences between them were expected (5) within the context of the match. Once the best matches are selected, the implied modifications (6) are made to the existing entity knowledge base (7), after being verified with the user (8), if necessary. Expectations of further modifications are generated from a variety of sources, including the information obtained from the discourse (9), the state of entities in the knowledge base (10), previously made modifications (11) and the state of the acquisition process.

The descriptions obtained from the expert must be presented to the matcher in the knowledge base's representation language. The purpose of the *discourse manager / user interface* is twofold: to permit a more "user-friendly" specification (e.g., natural language, graphics, menus, etc.) of these descriptions, and to provide K^n_{Ac} with any available cues as to the state of the discourse. Currently, the natural language protocols are translated, by hand, into the system's representation language. Discourse cues are assumed to be minimal, such as "topic" information.

Thus, the discourse fragment presented in Section 1 translates, approximately, into the structures shown in Figure 3. These structures may then be compared with selected entities from the existing knowledge base.

To avoid having to examine the entire knowledge base in order to assimilate the new information, entities that are most likely to be modified are selected as candidate matches. Thus, if there exists an expectation of some modification to a given entity description, that entity is compared to the new information from the expert. The way in which these modifications are anticipated will become clearer in Section 4. Initially, the only expectations available are based on the discourse cue recognizing that TRAVEL is a topic of interest. Hence, the entities semantically close to TRAVEL in the knowledge base are selected as candidate matches. These entities include TAKE-A-TRIP-AND-GET-PAID, shown in Figure 1.

The system then compares the expert's descriptions with the selected match candidates. The matching process, described more fully in Section 3, determines the similarities and differences between a pair of entity descriptions. The results of these comparisons are then evaluated in order to select the best match for each of the expert's entity descriptions. Section 3.2 describes the evaluation process, which rates the match results on a field-by-field basis and combines these ratings to produce an overall rating for each match.

For example, when the task described by the expert, i.e., EVENT-1, is compared with the existing task description TAKE-A-TRIP-AND-GET-PAID, the contents of each

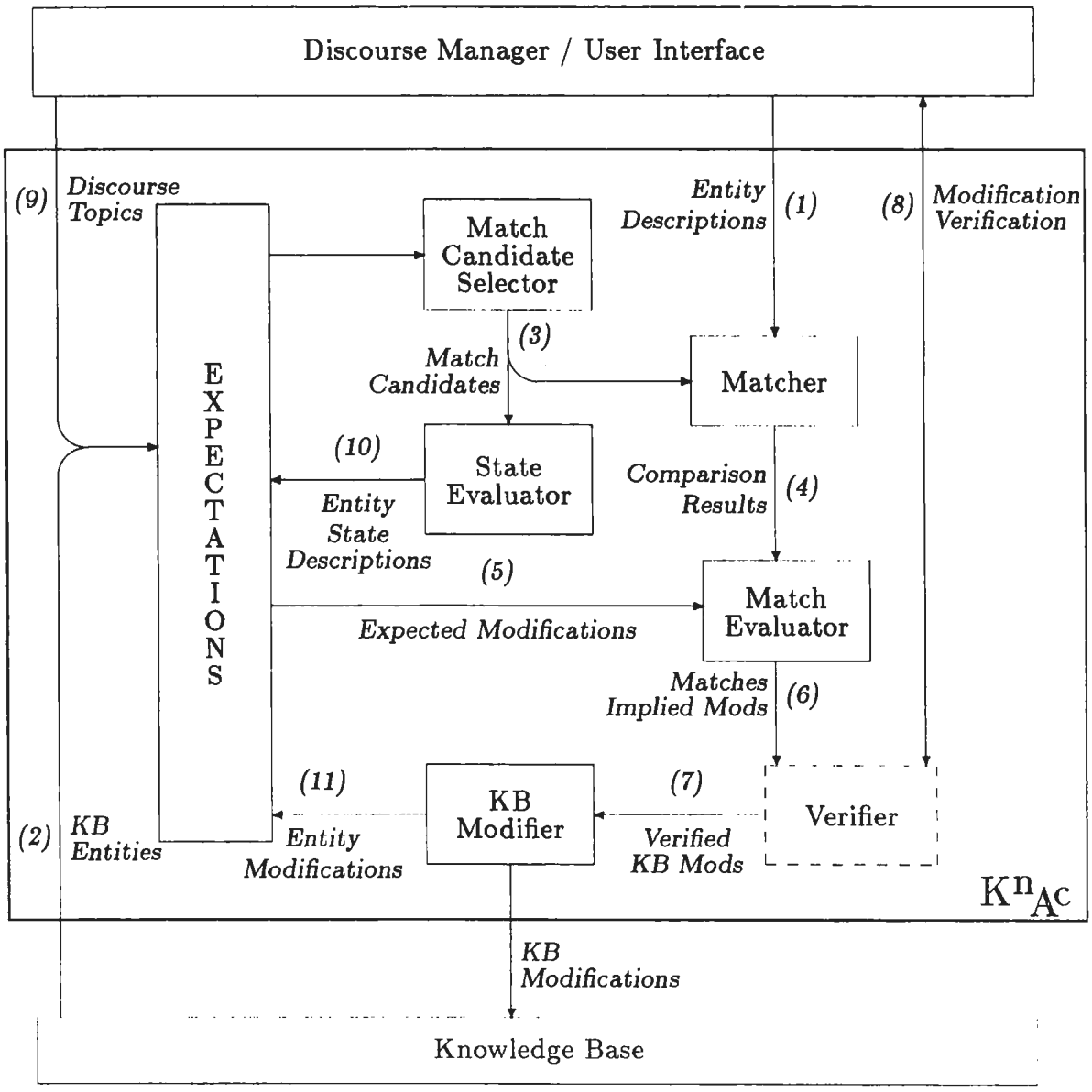


Figure 2: The K^n_{Ac} System Architecture

EVENT EVENT-1
STEPS: (ISSUE-TRAVEL-AUTHORIZATION TAKE-A-TRIP)
TEMPORAL-RELATIONSHIPS:
((ISSUE-TRAVEL-AUTHORIZATION *before* TAKE-A-TRIP))
CONSTRAINTS: ((DESTINATION *outside-of* STATE))
ATTRIBUTES: ((TRAVELER ···) (DESTINATION ···))

EVENT ISSUE-TRAVEL-AUTHORIZATION

EVENT TAKE-A-TRIP

OBJECT TRAVEL-AUTHORIZATION

Figure 3: Discourse Manager Output

field of these structures (e.g., *parts*, *generalizations*, *temporal-relationships*, *pre-* and *post-conditions* etc.) are compared. In the *steps* field, they have one entity in common (TAKE-A-TRIP) and each has one entity not found in the other (ISSUE-TRAVEL-AUTHORIZATION in EVENT-1 and GET-REIMBURSED in TAKE-A-TRIP-AND-GET-PAID). They have several *attributes* in common (TRAVELER and DESTINATION). Their *temporal-relationships* are mutually consistent, though different. Both entities are *specializations* of EVENT.

The ratings for these field matches is shown in Figure 5. Remember, these ratings reflect not only the degree to which the fields match, but more importantly (from the point of view of knowledge acquisition) the likelihood of the modifications, in future parts of the dialog, required to make them match.

From these ratings, the best matches are selected. If there is significant ambiguity, the matches are verified with the expert. If there are no acceptably good matches for one of the expert's entity descriptions, a new entity is added to the knowledge base.

When the best matches have been selected, the differences between the expert's description and the existing one are used to modify the knowledge base. For instance, the "extra" step in EVENT-1, i.e., ISSUE-TRAVEL-AUTHORIZATION, is added as a step of TAKE-A-TRIP-AND-GET-PAID. The extent to which this modification requires confirmation from the expert depends on the level of autonomy granted to the system. At various levels, all such modifications could be verified with the expert or only unexpected ones or only deletions, etc.

Once the knowledge base has been modified, new expectations are generated to be used in interpreting the next "discourse frame". The generation and management of these expectations is described in Section 4.

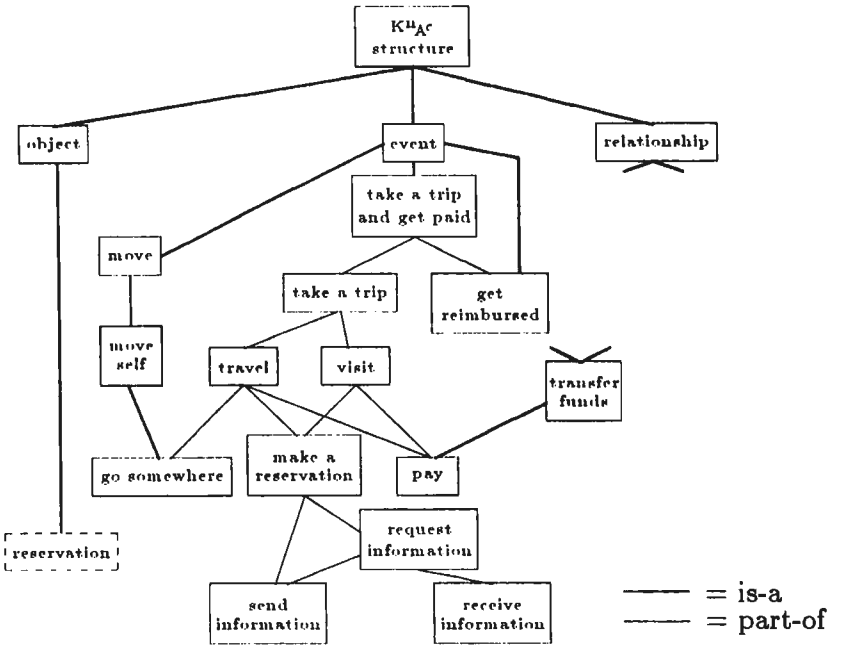


Figure 4: A portion of the knowledge base

EVENT-1 vs. TAKE-A-TRIP-AND-GET-PAID

Field	Rating
generalizations	1.000
parts	0.133
attribute-names	0.055
constraints	0.007
Match Rating	0.299

Figure 5: Ratings for field matches

3 Matching for Knowledge Acquisition

In order to match entity descriptions provided by the domain expert to those already known to the system, K^n_{Ac} must be able to compare these structures and evaluate the results. This matching process, while in some ways similar to that found in most recognition/interpretation systems, displays certain characteristics unique to knowledge acquisition. In particular, since the goal of a knowledge acquisition dialog is the modification of the knowledge base, the information provided by the domain expert should not completely match the existing entity descriptions. The matching process must be modified so as to be able to recognize and, where possible, anticipate these discrepancies. The matching techniques presented in this section make these discrepancies

explicit; the evaluation of these match results, described in Section 3.2, incorporates the extent to which these discrepancies were anticipated into its rating procedure.

3.1 Matching Entity Descriptions

POISE's knowledge base is represented in a frame-based language, similar to that used by systems like Knowledge Craft®[Wright and Fox, 1983,Carnegie Group, 1986]. Comparison of these knowledge structures requires matching on a field-by-field basis. Each field may be considered to be one of two types of structures: *sets of elements* such as *steps* and *generalizations* of an event, or *collections of constraints* such as the *temporal relationships* and the *pre-* and *post-conditions*. K^n_{AC} contains matching techniques for both of these types of structures.

3.1.1 Set Matching Determining how well two sets of elements match is not difficult; neither is determining how "different" they are (e.g., see [Tversky, 1977]). For knowledge acquisition purposes, however, the relevant question is "How likely is it that they can be modified so as to match?" To determine this, K^n_{AC} examines not only the elements in each (set) field of the knowledge structure, but also makes use of information *about* that field (i.e., *meta-information* or *facets*). In particular, information about the size of each field and the range of the elements in that field permit K^n_{AC} to calculate the probability that the *extra* elements in one of the structures will be added to the other.

Consider the comparison of the *steps* of EVENT-1 and TAKE-A-TRIP-AND-GET-PAID. A typical measure of similarity is:

$$\text{match-rating} = \begin{cases} 1 & \text{if } \text{Set}_1 = \text{Set}_2 = \emptyset \\ \frac{|\text{Set}_1 \cap \text{Set}_2|}{|\text{Set}_1 \cup \text{Set}_2|} & \text{otherwise.} \end{cases}$$

With one step out of the three unique steps between them in common, the similarity of these fields would be 1/3. How likely is it, however, that the "extra" step in the expert's description (i.e., ISSUE-TRAVEL-AUTHORIZATION) will be added to the existing description? Without requiring a deep understanding of domain-specific semantics, some additional information can still be used. If events usually have few steps (as specified by the meta-information about the *step* field of EVENT), the addition of this particular step is less likely than if there are many more steps still to be added. Similarly, if a step is going to be added, the range of possible steps will affect the probability of the desired one being added. This range is determined by combining the "type restriction" meta-information about the slot (i.e., a *step* of an EVENT must be

an EVENT) with the existing knowledge base (i.e., the number of EVENT descriptions known to the system).

Thus, K^n_{Ac} rates each set match based on the likelihood of the modifications implied by the match. The likelihood is determined from the anticipated size and range of the sets and requires no additional semantic information about their content. The derivation of these ratings is fully described in [Lefkowitz, 1987].

3.1.2 Constraint Matching Although comparing (or combining) arbitrarily constrained sets of entities is a difficult problem often requiring substantial domain knowledge, K^n_{Ac} compares sets of constraints, pairwise related by a common relation, in a purely mechanical fashion.[2] This section describes a mechanism for comparing such constrained sets; the mechanism is independent of the particular relation, requiring only a description of its algebraic properties, i.e., whether or not the relation is *reflexive*, *symmetric* and/or *transitive*.

First, any implicit constraints are made explicit by propagating the specified constraints using the relation's algebraic properties. The temporal relation *before*, for example, is only transitive; if the constraints (A *before* B) and (B *before* C) were specified, then (A *before* C) could be deduced. When the constraints are thus propagated, inconsistencies may be detected by check the results for any of the prohibited properties of the relation (i.e., non-reflexive, non-symmetric and non-transitive). If each set of constraints is internally consistent, the two sets may be merged and re-propagated, and this combined set of constraints may be checked for consistency.

If two sets of constraints are mutually consistent, a measure of their similarity may be obtained by determining the changes required to make them equivalent. Simply adding each set of constraints to the other would accomplish this, but this may add more information than is necessary. For instance, if the first set of constraints contained (A *before* B) and (A *before* C) and the second sets contained (B *before* C), then only (A *before* B) need be added to the second set. Requiring the addition of both constraints from the first set would imply a larger discrepancy between the sets than actually exists. Obtaining the minimal set of constraints that need to be added is not a trivial problem. K^n_{Ac} contains a new approach to generating these sets, called "opensures"[3], based on their algebraic properties.

Thus, as with fields containing *sets*, K^n_{Ac} is able to rate *constraint* matches by deter-

[2]The current system checks each relation separately; it does not handle interaction between different relations, though it is able to combine relations with their inverses. For instance, *before* and *after* constraints are handled together.

[3]i.e., the inverse of "closures"

mining the likelihood of the implied modifications. This rating is based on the algebraic properties of the relationships involved and requires no additional semantic information. K^n_{AC}'s methods for constraint propagation, determination of consistency, and generation of opensures are presented in [Lefkowitz, 1987].

3.2 Match Evaluation

After comparing each of the entity descriptions provided by the domain expert against those candidate entities selected from the existing knowledge base, K^n_{AC} evaluates these match results in two passes. First, the likelihood of two entities matching, based on the extent to which they differ and the probability of these differences being corrected, is determined as described above. This "degree of fit" is a relatively inexpensive means of pruning the set of possible matches.

The second pass of the match evaluation takes into account the context in which the comparison is being made. In addition to *how* the descriptions differ, it considers whether these differences are expected in a particular situation. The extent to which the modifications implied by the differences between the structures are expected serves as a "context-dependent" measure of the match. The anticipation of such modifications is a crucial part of the K^n_{AC} system and is described in the following section.

Consider, for example, the addition of the step ISSUE-TRAVEL-AUTHORIZATION to the description of TAKE-A-TRIP-AND-GET-PAID. The addition of such a step could have been foreseen for several reasons. First, since there were fewer steps in the existing description than are typically found in events, adding another step was reasonable. More importantly, upon examining the existing description to see if it was consistent and complete, it was discovered that a *precondition* of the step GET-REIMBURSED, describing the traveler as the recipient of funds, could not be satisfied by the only earlier step in the task (i.e., TAKE-A-TRIP). This further supported the addition of another step (occuring before GET-REIMBURSED) to the task.

4 ANTICIPATING MODIFICATIONS

K^n_{AC} provides a context in which to interpret information provided by a domain expert by anticipating modifications to an existing knowledge base. These anticipated modifications, or *expectations*, are derived from K^n_{AC}'s heuristic information about the knowledge acquisition process. This section describes how these expectations are generated, how they are used to provide a context in which matches may be evaluated, and how they ranked and managed.

4.1 Generating Expectations

K^n_{Ac} contains a body of heuristics about the knowledge acquisition process. These heuristics, obtained through the analysis of several knowledge acquisition dialogs, allow K^n_{Ac} to anticipate modifications to an existing knowledge base. These heuristics may be divided into four categories. The first group is based on the state of the knowledge, both that already contained in the knowledge base and new information provided by the expert. The second category depends on modifications previously made to the existing knowledge base. The third set makes use of a model of the discourse process, while the final set incorporates knowledge about teaching and learning strategies.

Since it is expected that the collection of heuristics will be modified, both as a result of improved understanding of the knowledge acquisition process and through the addition of domain specific heuristics, K^n_{Ac} allows heuristics to be added (or removed) in a straight-forward way. Most of K^n_{Ac}'s current heuristics are quite simple and domain-independent; the addition of more complex, application-specific heuristics may improve the system's performance in certain domains.

Consider a simple heuristic based on the state of an entity description:

Heuristic S2: *Fields with too few components will be augmented.*

This heuristic states that if information is detected to be missing, the addition of that information may be expected. Missing information may be detected in various ways. One simple approach compares the number of entries in a field of a knowledge structure with the field's expected cardinality. If the field is determined to contain too few values, additional values (of the appropriate type) will be expected. The expected field size may come, in order of specificity, from meta-information about a given field of a given entity, via inheritance from a generalization of the entity, from the default information for the type of the entity, or from an overall field default size. This size information may be static or determined dynamically by the system. An expectation generated by this heuristic is:

```
Exp146:  Expecting (certainty 0.360):
   MOD:  ADD ?New-part<is-a-knac-structure-p> to the
         Parts field of Take_a_trip_and_get_paid
         Derived from Take_a_trip_and_get_paid and H_S2.
```

Other heuristics based on the state of the knowledge exploit references to unknown entities, unsatisfied preconditions, and range/value conflicts to anticipate changes.

Changes to the knowledge base may imply additional modifications. For instance, two heuristics that are triggered when a new entity description is added are:

Heuristic M1: *Detailed information usually follows the introduction of a new entity.*

Heuristic M2: *Context information usually follows the introduction of a new entity.*

Additions of information to specific fields of entity descriptions (e.g., attributes, steps, constraints, etc.) form the basis of other modification heuristics.

Cues from the discourse manager, such as the topic of discourse or recently referred to entities, are the key to $K^n_A c$'s discourse heuristics. Because the current system lacks a sophisticated discourse manager, the only discourse heuristics being used are:

Heuristic D1: *Entities close to specified topics are likely to be referenced or modified.*

Heuristic D2: *Referenced entities are likely to be modified or referenced again.*

These heuristics generate expectations of some unspecified modification to the referenced entities or to those semantically close to them. "Closeness" is determined by the number and types of relationships (i.e., links) separating two entities.

4.2 Managing Expectations

As the number of expected modifications to the knowledge base grows, $K^n_A c$'s ability to use the expectations to focus its attention diminishes. Thus, a means of selecting the most likely expectations (for a given time) is required. This is accomplished by assigning a rating to each expectation and pruning the set of heuristics based on this rating.

Each heuristic is responsible for determining a rating for each expectation it generates. This rating depends on the quality of the data and the specificity of the heuristic. In addition, each heuristic assigns a function to its expectations that specify how these ratings will change with time. For instance, certain expectations are important when they are created but become less valid with the passage of time; others become more critical as time passes. Some become more (or less) significant based on some state of the knowledge base; others are always valid. The functions currently available in $K^n_A c$ are: *fade*, *increase*, *until*, *while*, *after*, *for*, *always* and *never*.

5 STATUS AND CONCLUSIONS

In this paper we have examined an often overlooked aspect of the knowledge acquisition process: the assimilation of information presented by a domain expert into an existing knowledge base. Though a fundamental part of the current conventional knowledge base development process, the issue of automatically locating and appropriately modifying existing knowledge to conform to the domain expert's descriptions has received little, if any, emphasis. Most current knowledge acquisition tools place this burden on the domain expert, forcing him to take over part of the knowledge engineer's task. By automating this assimilation process, the K^nAc system better insulates its user from the knowledge base.

K^nAc accomplishes this assimilation by: 1) comparing entity descriptions provided by the domain expert with existing knowledge base descriptions, 2) evaluating these matches in the context of the knowledge acquisition discourse, 3) making the modifications to the existing descriptions implied by the expert's information, and 4) generating (and managing) expectations of further changes to the knowledge base.

This work has developed several generic matching (and match evaluation) techniques especially adapted for knowledge acquisition. They shift the focus of matching from examining how closely two entities match to exploring the likelihood of their being modified so as to match. This is accomplished by matching procedures (for sets of entities and collections of constraints) which determine differences as well as similarities, an evaluation technique which explores the probability of the modifications required to make entities match and the degree to which these modifications are expected, and a means of anticipating modifications to the knowledge based on heuristic information about the knowledge acquisition process.

The K^nAc system is implemented in Common Lisp running on a TI Explorer®. Its current use is still experimental, though it has been able to assimilate a 20 step dialog on the travel reimbursement process, correctly constructing an internal representation of the plan in the POISE knowledge base. A complete description of the implementation and the sample dialog can be found in [Lefkowitz, 1987].

References

[Carnegie Group, 1986] *Knowledge Craft Users Manual.* Carnegie Group Inc. (1986).

[Croft and Lefkowitz, 1984] Croft, W.B. and Lefkowitz, L.S. Task support in an office system. (1984) *ACM Transactions on Office Information Systems*, 2(3).

[Croft *et al.*, 1982] Croft, W.B., Lefkowitz, L.S., Lesser, V.R., and Huff, K. POISE: An intelligent assistant for profession based systems. (1982) In *Proceedings of the Conference on Artificial Intelligence*, Oakland University, Michigan.

[Eshelman *et al.*, 1986] Eshelman, L., Ehret, D., McDermott, J., and Tan, M. MOLE: A tenacious knowledge acquisition tool. (1986) In *Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta, Canada.

[Ginsberg *et al.*, 1985] Ginsberg, A., Weiss, S., and Politakis, P. SEEK2: A generalized approach to automatic knowledge base refinement. (1985) In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 367–374.

[Kahn *et al.*, 1985] Kahn, G., Nowlan, S., and McDermott, J. MORE: An intelligent knowledge acquisition tool. (1985) In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 581–584.

[Lefkowitz, 1987] Lefkowitz, L.S. *Knowledge Acquisition through Anticipation of Modifications.* (1987) PhD thesis, University of Massachusetts, Amherst, MA.

[Tversky, 1977] Tversky, A. Features of similarity. (1977) *Psychological Review*, 84(4):327–352.

[Wright and Fox, 1983] Wright, M. and Fox, M.S. *SRL 1.5 User Manual.* (1983) Intelligent Systems Laboratory, Carnegie-Mellon University Robotics Institute.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

Cognitive Primitives

Alain T. Rappaport
Neuron Data 444 High St Palo Alto CA 94301.
(The Robotics Institute, Carnegie-Mellon University, Pittsburgh PA 15213)

ABSTRACT: This paper addresses the problem of the level of abstraction at which knowledge based system primitives must be developed so as to facilitate knowledge acquisition. Low level programming or the use of task-level methodologies as they exist either prevent a rapid learning and development or lock the developer in a given problem-solving paradigm. We explore, with examples, the principles underlying the design of a compromise-level set of cognitive primitives. The latter can then be used to generate higher, task- or domain-level functions. Examples are presented to illustrate this approach. The role of experimentation in conjunction with theory is emphasized in the design of tools which mtach more closely the human expert knowledge, difficult to decompile and thus to represent in more classic formalisms.

INTRODUCTION

A spectrum of different approaches to knowledge acquisition for knowledge based systems has been developed, ranging from domain knowledge analysis (Boose 1985, Gaines and Shaw 1986) to task-dependent shells (Eshelman and McDermott 1986) and constraint monitoring (Marcus 1986) via apprentice learning techniques (Wilkins 1986). Other approaches have also been suggested (Carbonell 1985). The present research is undertaken in the context of the development and use of the NEXPERT *OBJECT* knowledge base building tool. It concerns the specification and design of knowledge based system primitives in relation to the knowledge acquisition task. Knowledge acquisition issues must be considered *prior* to the design of the tool. The issues involved range from the nature of the primitives and the type of control structures to the development interface.

KNOWLEDGE DESIGN: LEARNING BY EXPERIMENTATION

Designing a knowledge base is itself a problem solving task (Rappaport 1986). The successful completion of a knowledge based application is dependent upon the *learnability* of the system's functions. At the beginning of the interaction, the only expressible goal of the expert is to build an effective application. Thus, the distance between the goal and the current state of the problem is too large to speak of a goal-driven behavior. No function can measure this distance and use it as a heuristic in the problem solving task. Hence, the behavior appears to be goal-free since it does not take into

account the applicability of the operators to the overall problem. This phase corresponds to a period of *learning by experimentation.*

The notion of experimentation requires the existence of a highly interactive environment where rapid feedback to the design actions is given to the developer. Experiments can be based on the use of structural analogies, reproducing examples or imitating. In order to progress towards more structured problem solving techniques, the examples or the functions which can be explored must be of a high enough level of abstraction, with some degree of psychological relevance. Learning by experimentation in this situation leads to the discovery of *operators* or *macro-operators* made of lower level primitives. This process is based on the identification of subgoals being reached by assembling elementary functions in a particular manner. The subgoals allow to relate the system's functionalities to subtasks of the overall problem and to start having a "metric" structure whereby the distance between a particular function and a cognitive function can be estimated and used in the problem solving. In other terms, the problem space becomes serializable for the domain expert.

As the domain theory (how to use the primitives in relation ot the application) is enriched in the expert's mind, the progression leads to the identification of useful operators or macro-operators (Cheng and Carbonell 1986), and higher level reasoning techniques such as plan instantiation (Wilensky 1983), reasoning by derivational analogies (Carbonell 1985). Such techniques are in fact used once learning by experience has occured in any domain.

In other terms, experimentation with the lower level primitives allows the construction of higher level, more complex functions, closer to either the task or the domain level. However, should the initial primitives be little meaningful in terms of psychological relevance, a great deal of programming effort will have to be made in order to reach a level of representation and control comprehensible by a domain expert. On the other hand, providing a domain expert with built-in functions at a high level restricts the creativity and the likelyhood of success.

Our approach has been to study and provide domain experts with primitives of a relatively high level of abstraction from which functions at the task or domain level can be constructed. The expected result is the definition of a new type of language in which, since they are built of lower level and yet psychologically relevant primitives, the task-level and domain-level functions are *editable.* This provides the designer with the critical capability of *jumping out* of pre-defined, often ill-fitted, computational paradigms. In the remaining sections, we emphasize some fundamental aspects of the design of cognitive primitives.

Mapping

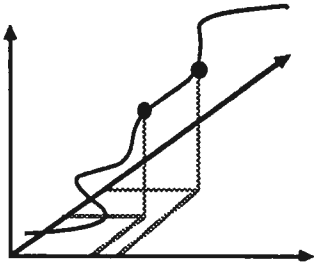
Let us assume that the concept *A* is expressible in the language *L1.* To express *A* in language *L2*, one must apply one or several *transforms.* The translation of a word from one language to another is such a transform. In simple cases, there exists a straightforward *mapping* relationship. A general, non-linguistic definition of the mapping process involves making a correspondence between a particular point in a world *W1* and another in a world *W2.* For instance, the mapping of a surface into a plane is an exercise where the *dimensions* of the two respective worlds are well defined. When the worlds in question are the human mind and the program, the definition of such mappings becomes less clear because of the ill-defined structures of the respective worlds.

The human expert.

We shall consider that *W1* is a human expert's mind. What characterizes a human expert is its *behavior.* Search has been largely compiled into knowledge, providing a subjective, powerful index to the book knowledge base of facts and rules to verify. The effect of using such compiled knowledge is naturally reflected in the *economy* of reasoning which differentiates the expert from the novice.

At any coordinate *t* along the time axis, a domain expert solving or structuring a problem is in an *n*-dimensional world. Because many of those dimensions are irrelevant to the problem solving task itself, we can restrict *Sr* to the set of relevant dimensions. Those are, for instance, the book knowledge involved or the intuitive chunks of knowledge as well as incoming data from the external world. Thus, at time t_0, the expert's state of mind can be described theoretically in terms of coordinates in those dimensions. We shall call this space the *cognitive phase space.*

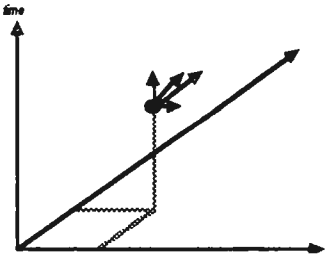
In a graphic representation, one axis is used to described the flow of time and two others to describe the cognitive phase space. The following figure shows the domain expert at time *t* and at time $t+\Delta t$.One naturally obtains a *trajectory* when joining such points:



Such a trajectory corresponds to each problem solving task . The very notion of trajectory suggests a *focus of attention* in the expert's mind. What is essential to consider now is what determines such trajectories. The design of intelligent systems should then be inspired from the findings at this high level.

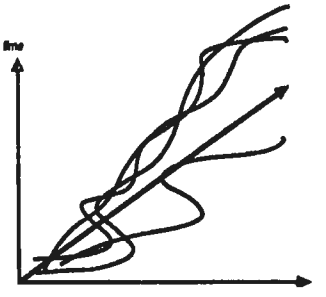
Working at this level unveils elementary and powerful principles. The first one we shall state is the *principle of the unfolding line.* The unfolding line reflects the conservation of the focus of attention, the main behavioral characteristic of an expert's behavior.

The unfolding line is a macroscopic representation of the problem solving activity. The reality is that each possible trajectory happens to be the resultant of many consecutive vectors whose direction is dependent not only upon internal knowledge but also upon information received from the outside world. The role of the internal knowledge is to direct the search, taking the external world information as well as the internal state into account ; it represents the boundaries in the cognitive phase space by constraining the set of potential directions, thereby conserving the focus of attention. The following figure illustrates this notion of local vector.



In this metaphor from physics, the vector representing the new direction of the focus of attention is represented in dark. It is the resultant of time and the two arbitrary axes representing the current cognitive state (external and internal factors). This vector is thus dependent upon the past history (which led to the present state) as well as the current external events.

If a general law governs the determination of the unfolding line, how can one account for the diversity of trajectories which can be observed for the same problem being solved by different experts?



Different experts can solve the same problem using different trajectories. One of them has solved it in much less time on this figure.

It is conceivable that at any given time along the trajectory, a very general reasoning principle accounts for the direction of the problem solving vector. However, if the principle is global in terms of its powerful effect on the reasoning direction, it is *local in time*. At some other time, the direction of the vector may be largely determined by the recognition of a significant detail in a metonymical process. Intuitive knowledge as well as goals suggested by external agents might intervene as well. In general, several solutions may exist at any given point in the reasoning process, either as a result of knowledge or as a function of external but related events.

Two derived principles can thus be expressed. Trajectories are not only determined globally but also locally, at the *microscopic level*. Furthermore, the notion of an unfolding line resulting from a microscopic description is consistent with that of an *event-driven* behavior.

The artificial formalism

Since we expect that an artificial formalism that it represents the expert's behavior faithfully in order to have the same efficiency, we may state that in order to perform the mapping between *W1* and *W2*, the primitives available in *W2* must allow one to reproduce to the best degree the observed trajectories of the human experts.

The more complex the mapping functions are between the human expert and the artificial formalism, the more difficult it will be to ensure that the artificial system follows an acceptable trajectory when it performs. The overall mapping effort might well become economically unsound. The mapping problem itself must be *serializable*; therefore there must exist some initial degree of *similarity* between the way the human expert expresses its knowledge and the available functions in the knowledge base building tool.

How does the notion of trajectory affect the definition of computational models? How does this hypothesis place constraints on and help specify the nature of *W2* on which the mapping must be done?

The mapping distance

In relation with the notions described above, we shall consider two problem spaces in order to formalize the mapping problem.

(i) ***M:*** the mental domain problem space, corresponding to the current mental representation and perception of the problem by the domain expert.

(ii) ***F:*** the tool functions problem space.

The key factor in being able to map in the second space operators or macro-operators relevant to the mental model in *M* is the existence of a matching distance between the two spaces.

M is the problem space of the task from the expert's viewpoint. It is the mental model of the task. To a certain extent, the expert is able to describe how his/her focus of attention varies, how certain situations are recognized and taken into account. S/he is able to naturally *derepresent* the task to a certain level. For example, the task can be represented as a hierarchy of subtasks or embedded sequences of reasonings, variations in the focus of attention, revisions or iterative cyclings. In most cases however, the level of granularity of these descriptions will not be *very* low, for it consists of compiled knowledge.

F is the space of elementary transforms or primitives which are available in the tool. Some primitives are arrangements of less complex ones, and therefore are more abstract. We shall detail those primitives later.

The notions of serializability of the mapping problem and of similarity between the operations performed in both worlds lead us to express the mapping problem in terms of minimizing a *distance* between the representations. Let *D(M,F)* be the matching distance between the two spaces. *D* is a measure of how the primitives in the mental space *M* can be represented by primitives in the system's space *F*. Minimizing *D* constitutes the principle of *minimal decompilation.*

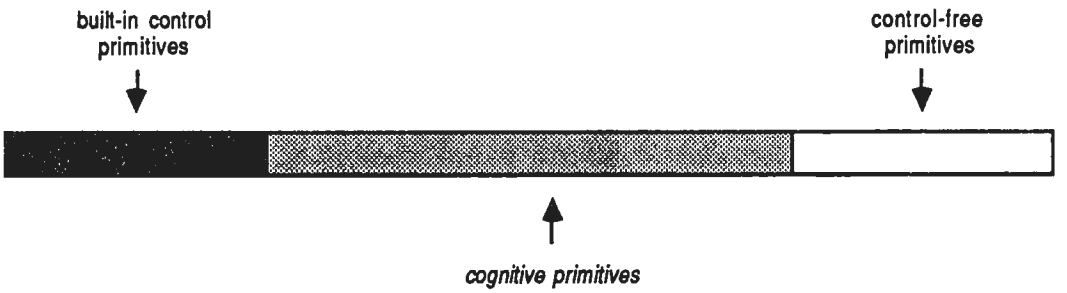
It is essential that the levels of abstraction of the primitives available in the tool allow a rapid development of the knowledge based application by matching as closely as possible the actual behavior of the expert. Because the design of such primitives is driven by the minimal decompilation principle and the principles described above, we call them *cognitive primitives.*

COGNITIVE PRIMITIVES

Control-free or very-few-control languages such as LISP and OPS5 do not minimize *D(M,F)*, unless much higher level structures have been built with them. In the former language, the primitives, as for any such language, are low-level, requiring relevant primitives to be built before the actual task is envisioned. When a rule is written it may have been considered by the expert as a cognitive unit. However, most efficient interpreters (Forgy 1977) will not at all treat it as such, thereby reducing the structural fit. The intention of the designer is not correctly represented because an intimate knowledge of the interpreter or some programming is necessary to achieve the intended result. The interpreter will perform *unexpected* tasks, from the cognitive viewpoint, such as a weak procedural conflict resolution and numerous global partial-matching cycles. A coherent focus of attention can only be preserved by programming around the basic algorithm.

At the other end of the spectrum, built-in control languages such as EMYCIN do minimize this distance but merely constitute *one* basic primitive or procedure, thereby strongly limiting the interest and applicability of the system. Prolog is a language which belongs to both ends of the spectrum: it is used to perform pure backward chaining or as a programming language.

The following figure shows the spectrum of languages along an axis representing the degree of control accessible to the knowledge designer.



By analogy, consider a drawing tool. For example, such dedicated systems would correspond to tools which primitives would be complex objects such as flowers. This is not a drawback if one needs to draw flowers; it is quite inadequate however to draw other items such as houses. Cognitive primitives are the equivalents of the Quickdraw procedures `FrameRect`, `FrameOval`, `FrameRoundRect`, `FillArc`, `UnionRect` etc... (Rose et al. 1985) in the domain of representing thought processes rather than images. It is quite easy to realize how such functions do minimize *D(M,F)* in the graphic domain.

Psychological relevance

The design of cognitive primitives takes into account the different principles stated above and the need to minimize *D(M,F)*. In other terms, cognitive primitives must have a basic *pychological relevance.*

The principle of the unfolding line means that the focus of attention of the domain expert must be preserved throughout the transformation. The same principle also suggests that at any point in time, the system should be in a local but meaningful state. Indeed, the notion of focus of attention introduces the notion of history. Thus, any point of reasoning has an origin and represents a meaningful entity.

The principle of determination of the trajectory at a *microscopic* level implies that the chunks of knowledge are more or less independent units which are able not only to test conditions but to alter control mechanisms and determine the next units to be explored. Control must be altered by local mechanisms as well as more general procedure; there is declarative access to the control mechanisms.

Since local points along a trajectory are *meaningful* in that they reflect a somewhat autonomous chunk of knowledge, units of representation (for example, rules) must be seen as small functional units, fully evaulated when selected. In conjunction with local control links, the relative autonomy (a rule could need another for its evaluation) implies a better control of the navigation in the knowledge graph.

Finally, the graph nature of the architecture is consistent with the *event-driven* architecture of a focus-based system. An interrupt will always be likely to occur during a well-identified task or subtask

corresponding to the current focus. Systems with interpreters largely dependent on the implementation rather than on cognitive consideration cannot have a proper event-driven mechanism, necessary to handle applications in reactive worlds.

There are other issues regarding, for instance, the nature of the *logic* used to compute such primitives. An obvious consequence of the mapping approach is to also approach the logic design from the psychological viewpoint. In that sense, the closed-world assumption seems fundamental.

Interactivity

Cognitive primitives cannot be designed without taking into account the interaction between the designer and the environment in which they are to be manipulated. If the access to the primitives and in particular the testing of the design is made difficult, the advantage of approach is obliterated by the lack of *usability* (Gaines 1986). Graphic displays must relect the internal trajectory, or the focus of attention, and representation in the system.

The interface and compilers must allow numerous design and refine cycles as they natually occur in knowledge acquisition and design. All levels of information and representation must be clearly expressed. Provided that one has fast, incremental compilers, the interface must make use of perceptual mechanism, in particular visual thinking (Arnheim 1967). In short, the system must ensure the possibility of learning by experimentation.

Decomposing the distance

We can in fact describe more precise measurements to take into account when designing such primitive in order to minimize the distance *D(M,F):*

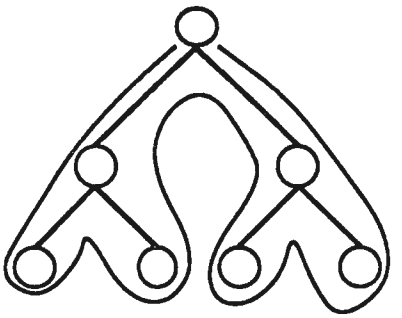
- ds: degree of structural comprehensiveness (how examples appear relevant to the domain expert in terms of any situation or action in his or her behavior at any given time)
- da: degree of autonomy (how much it can be used independently of the general control structure)
- di: degree of interpretability (how rapidly can it be tested and evaluated in the current architecture).

In the following sections, we explore further the notion of cognitive primitives through some examples and notes from case studies using the NEXPERT *OBJECT* tool .

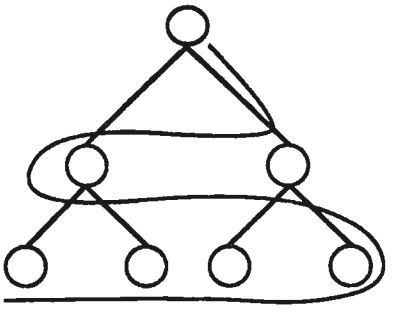
We shall see that other criteria arise in the definition of such primitives such as their ability to be used in conjunction and to form higher level functions, closer to the task and domain levels.

The *context* as an unfolding primitive

The search in a tree of nodes representing a diagnostic procedure can be performed in many different ways including the most simple depth-first and breadth-first. However, the actual account of the search by a domain expert will follow a *best-first* strategy, which sometimes exploits more intuitive, experiential heuristics than any other rule.

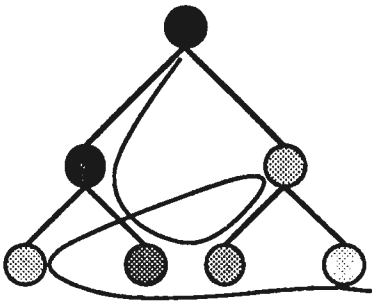


a simple depth-first search



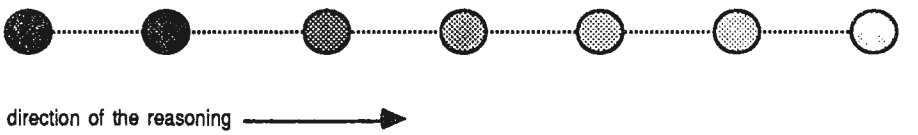
a simple bredth-first search

The *best-first* strategy necessarily reflects the expert's behavior. It is critical to capture this expression of the reasoning trajectory. Thus, a primitive which *linearises* the search consitutes a powerful and economic representation of the control, as opposed to relying on a lower-level language to edit basic strategies such as breadth- or depth-first or any other pre-determined search strategies.



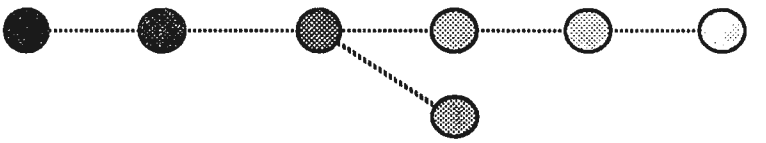
a best-first search

The "context" primitive allows one to express such a search in the actual order. The result is a very flexible linear structure:

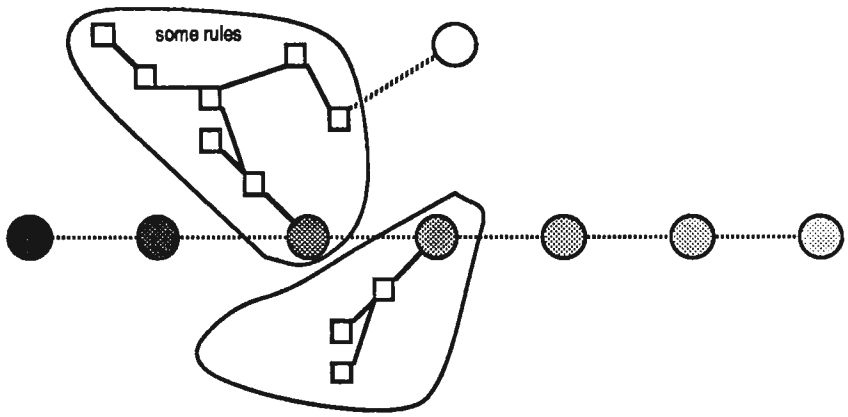


direction of the reasoning

The primitive matches very closely the notion of focus of attention. Its flexibility allows the developer to work with and edit very abstract knowledge. It is for instance very easy to create the following structure:



The primitive represents what we call a *weak link*. Since one can simply focus on another hypothesis, one does not need to express the reasons as is the case in the tree structure made of *strong links*.
The context introduces the concept of "knowledge island". Indeed, each node may require a very specific evaluation reflecting a "deeper" level of knowledge:



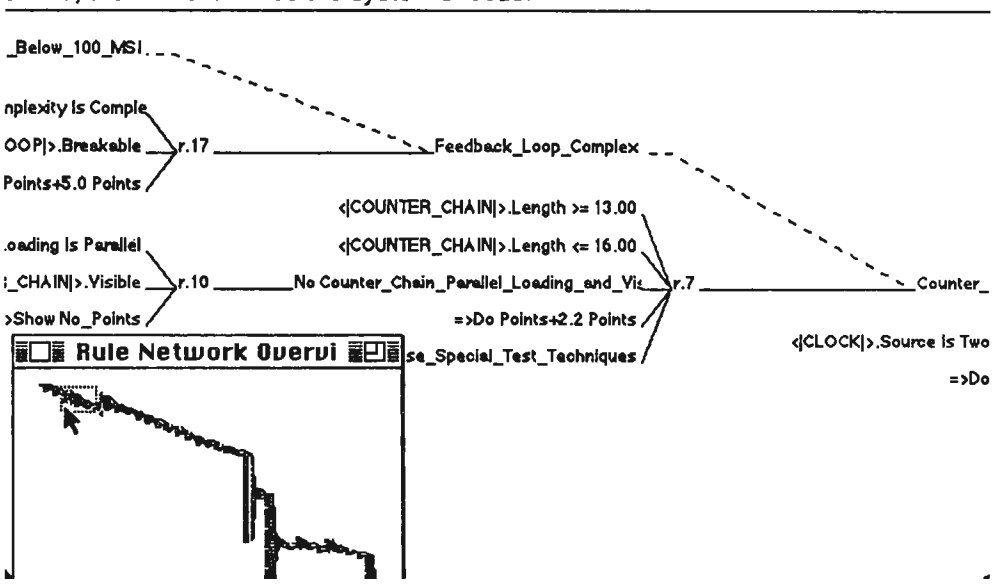
a knowledge island

A complete architecture is thus progressively designed. The primitives with different levels of abstraction such as the context allow the design of the application following the human specification. Large systems such as R1 (McDermott 1982) have been under development and use for a long time and now have structures similar to the one above although the underlying algorithm does not match such structures which thus require much reworking.

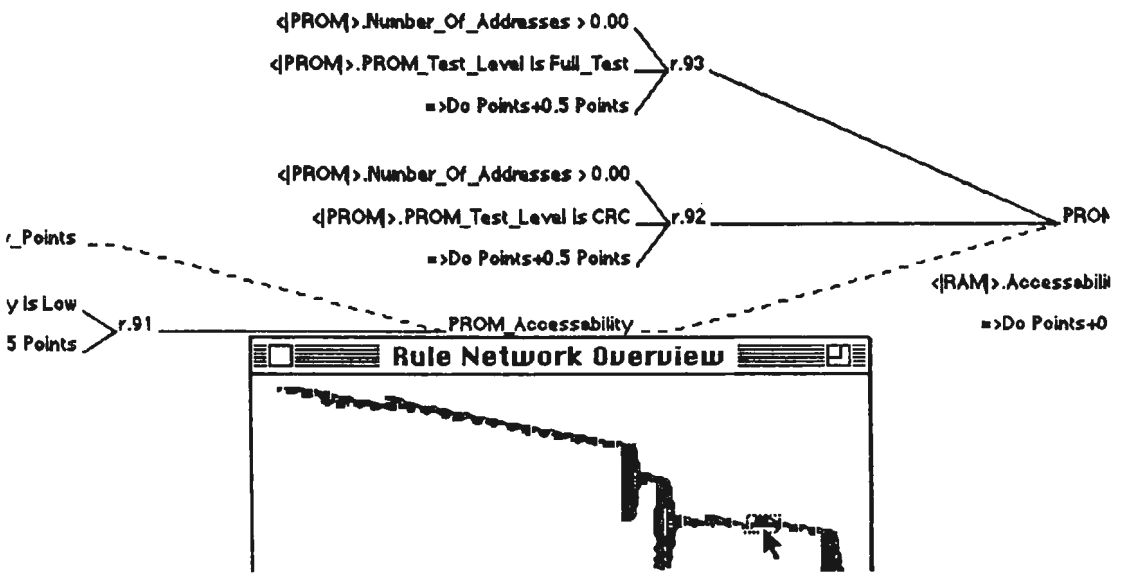
The next figure shows a trivial example of knowledge islands made by an expert using the context primitive for a verification application.



The following figure illustrates the use, by the same expert, of an analogical high level control mechanism in a "Card Design Assistant" application to insure the testability of custom electronic cards. In this case also, the context drives the system's focus.



The Rule Network Overview window shows the overall *unfolding line* architecture of the knowledge base. The rectangle indicates the location of the larger display in the background where knowledge islands are separated by dotted lines.

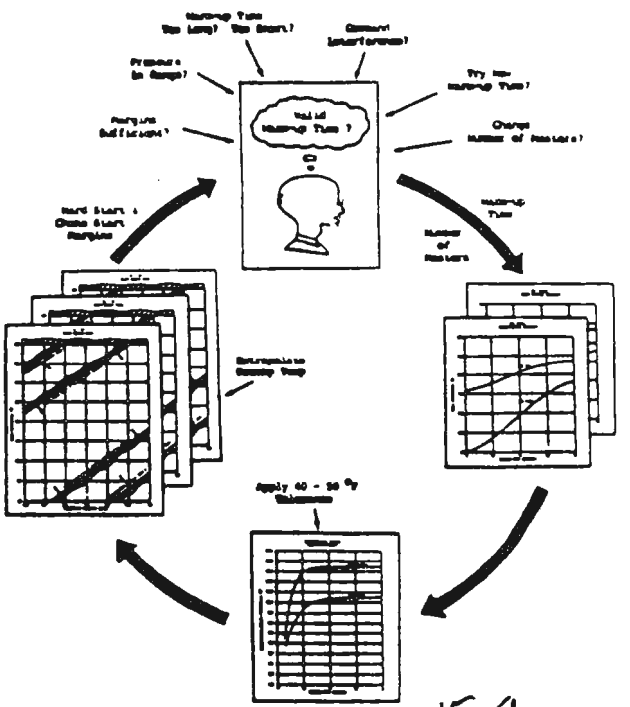


Another area of the same knowledge base is visualized on this figure.

The description of the context primitive also emphasizes the flexibility required in considering the use of methodologies as different as backward and forward chainings. From a tree-like, backward structure difficult to edit one has shifted naturally to a graph structure. The exploration of the graph can be controlled by various means, at the *local* as well as *global* levels. The problem is decomposed into cognitively relevant chunks or associations of chunks of knowledge.

Building an iterative task

Another case study is the development (and delivery) of an application accomplished in four man-weeks of engineering efforts, by a domain expert "virtually void of experience in expert systems application". (Robinson and Satterlee 1987)



The previous figure shows a diagram from the domain expert, illustrating the task (Robinson and Satterlee 1987). The application replaces "a complex iterative process previously performed by the spacecraft operations staff", namely the configuration of thrusters on orbiting Intelsat spacecraft.

Different factors seem to have been critical in achieving the application, ranging from the nature of the primitives to the interactiveness of the system, in order to "emulate the human reasoning process" (Robinson and Satterlee 1987).

- "non-monotonic control primitive of re-evaluating previous inferences automatically as changes occur in the data-base". An example is given of a rule performing the following actions:

- "ability to accommodate the external mathematical expressions".

- " the use of plain English statement structure in rule formulation".

- "a NEXPERT knowledge-based system is functional at any stageof development, since it takes only one rule to constitute a runnable expert system". This comment reflects the important role of the interactivity. Since each rule fits in a particular place in the architecture designed , it can be tested (interpreted) immediately.

- "the useful feature of displaying the reasoning path by which the inference tool arrived at its answer". The graphic interface takes full advantage of the role and selectivity of visual thinking in the human tasks (Arnheim 1969).

The iterative task was modeled without a task-level frame, but rather by progressively assembling chunks of knowledge in a graph matching the human task. The iterative behavior stems from the use of a *revision* primitive which allows the systems to detect changes in LHS conditions states and accordingly redirect the inference.

COGNITIVE PRIMITIVES AND TASK-LEVEL FUNCTIONS

Task-level and domain-dependent functions are however necessary to provide enough guidance in knowledge design. We suggest however that such functions must be expressible in terms of lower level cognitive primitives so that they do not constrain the developer in fixed computational paradigms. Because the task level primitives should thus be editable , it must be possible to jump out of their framework.

Generic tasks (Bylander et al. 1986) constitute frameworks which, in our perspective, stand near the built-in control end of the spectrum. Each task is limited by one or two approaches. Even if debugging facilities are provided the developer has to remain within the same problem solving methodology. Simple tasks such as diagnostics and classification could be performed in many different ways, using for instance either forward chaining or hypothetical reasoning instead of more classical techniques.

Another approach has been to create task-level templates to use languages which are not necessarily well adapted to the task itself. The drawback is also the lack of flexibility of the template itself.

If task-level operators are designed from primitives at the programming level, then the potential for their editing in case of a mismatch (local or global) is considerably reduced. On the other hand, if they are built

out of comprehensive primitives in terms of control and representation with which one can express many different types of reasoning, then the flexibility is optimal. By the same token, an open architecture must allow flexibility at the level of the primitives themselves.

A THEORETICAL AND EMPIRICAL RESEARCH

The design of both the cognitive primitives and the higher level primitives or functions at the task-level will rely on two *indissociable* approaches: (i) the theoretical, taking into account cognitive psychology as well as more recent research in artificial intelligence and (ii) the empirical research which consists in providing the results of the theory as quickly as possible to the actual holders of the knowledge, the domain experts and integrating the feedback in the fundamental design. Each primitive can then be either validated or show not to be useful, or restricted to more specific situations. It can be found to be a key function to implement a higher level task, such as the revision function in the iterative tasks. The identification of those task level properties of such primitives (which do not only concern reasoning but also objects and inheritance) through their use in different domains should lead to new, flexible tools, both task-independent and -specific.

THE COGNITIVE TOOLBOX CONCEPT

The concept of a cognitive toolbox is to provide a development environment following the design principles emphasized above. Along with a highly interactive interface which will make the knowledge acquisition phase benefit from the developer's perceptual intelligence, the primitives offered in the system must remain as close as possible to the mental representations of the reasoning functions to be formalized. This is again the principle of minimal decompilation.

Such a toolbox constitues a different and complementary approach to knowledge acquisition which relies more on a domain-general perspective. Most important, this approach emphasizes the role of the domain expert in the knowledge design process, as other highly interactive techniques do (Boose and Bradshaw 1986; Kitto and Boose 1986) and the need to design tools requiring less *deconstruction* of the mental representation of the task, closer to the task-level (Gruber and Cohen 1986). Different methodologies will also capture different types of knowledge (Gaines 1986), from the more search-intensive, algorithmic, to the more compiled, experienced-based human knowledge, and therefore should coexist.

The present research aims at a comprehensive theory of the computational functions which represent the best equilibrium between algorithm- or implementation-driven systems and fixed task-level methodologies. The use of a largely fielded system in many different domains and tasks has been most helpful in this respect. Furthermore, the identification of a toolbox of powerful cognitive techniques to represent expert knowledge should have interesting consequences for other mechanisms such as learning, inevitably inherent to knowledge-based agents.

Acknowledgements: The author thanks Jaime Carbonell for his comments and discussion of the present work as well Greg Hood and Bruno Jouhier for their most useful comments. Thanks are extended to the domain experts who have provided reports of their work for the examples reported here.

References

Anderson JA (1984) Learning to program LISP, Cognitive Science 8:87-129

Boose JH (1985) A PCP-based method for building expert systems, Boeing Computer Services Report BCS-G2010-24

Boose JH (1986) Expertise transfer and complex problems: using Aquinas as a knowledge acquisition workbench for expert systems, *in* Proceedings of the first international workshop on knowledge acquisition for knowledge-based systems, AAAI, Banff, Canada.

Bylander T and Chandrasekaran B (1986) Generic tasks for knowledge-bases reasoning: The "right" level of abstraction for knowledge acquisition, *in* Proceedings of the first international workshop on knowledge acquisition for knowledge-based systems, AAAI, Banff, Canada.

Carbonell JG (1982) Learning by analogy: formulating and generalizing plans from past experience, *in* Machine Learning :137-160, Morgan Kaufmann, Los Altos CA.

Carbonell JG (1985) Derivational Analogy: atheory of reconstructive problem solving and expertise acquisition, *in* Machine Learning II :371-392, Morgan Kaufmann, Los Altos CA.

Cheng P and Carbonell JG (1986) The FERMI system: Inducing Iterative Macro-operators from Experience, Proceedings of the Fifth National Conference on Artificial Intelligence :490-495, Morgan Kaufmann, Los Altos CA.

Ericsson KA and Simon HA (1984) Protocol Analysis, MIT Press, Cambridge MA.

Eshelman L and McDermott J (1986) MOLE: a knowledge acquisition tool that uses its head, Proceedings of the Fifth National Conference on Artificial Intelligence :950-955, Morgan Kaufmann, Los Altos CA.

Gaines B (1986) An overview of knowledge acquisition and transfer, *in* Proceedings of the first international workshop on knowledge acquisition for knowledge-based systems, AAAI, Banff, Canada.

Gaines B (1986) Designing expert systems for usability, *in* Human Factors for Informatics Usability, Schakel B (ed).

Gruber T and Cohen P (1986) Design for acquisition: principles of knowledge system design to facilitate knowledge acquisition, *in* Proceedings of the first international workshop on knowledge acquisition for knowledge-based systems, AAAI, Banff, Canada.

Gupta A (1984) Parallelism in production systems: the sources of expected speed-up, Technical Report, Carnegie-Mellon University, Dept. Computer Science CMU-CS-84-169.

Kant E (1985) Understanding and automating algorithm design, Proceedings of the Ninth International Joint Conference on Artificial Intelligence 2:1243-53, Morgan Kaufmann, Los Altos CA.

Kitto C and Boose J (1986) Heuristics for Expertise Transfer: an implementation of a dialog manager for knowlege

acquisition, in Proceedings of the first international workshop on knowledge acquisition for knowledge-based systems, AAAI, Banff, Canada.

Marcus S (1986) Taking backtracking with a grain of SALT, *in* Proceedings of the first international workshop on knowledge acquisition for knowledge-based systems, AAAI, Banff, Canada.

McDermott J (1982) R1: a rule-based configurer of computer systems, Artificial Intelligence 19:38-88.

Nilsson N (1980) Principles of Artificial Intelligence, Tioga Publishing Company.

Rappaport A (1986) Goal-free learning by analogy, *in* Machine Learning: a guide to current perspectives :261-267, Kluwer Academic, Boston MA.

Rappaport A (1986) Multiple problem spaces in the knowledge design process, *in* Proceedings of the first international workshop on knowledge acquisition for knowledge-based systems, AAAI, Banff, Canada.

Robinson JAand Satterlee AA (1987) An in-service expert system for configuring orbiting spacecraft thrusters, submitted for publication.

Rose C et al. (1985) Inside Macintosh, Addison Wesley, Reading MA.

Rosenbloom PS and Newell A (1985) The Chunking of goal hierearchies, *in* Machine Learning II :247-288, Morgan Kaufmann, Los Altos CA.

Wilkins DC (1986) Knowledge base debugging using apprenticeship learning techniques, *in* Proceedings of the first international workshop on knowledge acquisition for knowledge-based systems, AAAI, Banff, Canada.

Wilensky R (1983) Planning and Understanding, Addison-Wesley, Los Altos CA.

The Problem of Intelligent Example Selection

Edwina L. Rissland [1]

Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

Abstract

In this paper, I discuss the problem of selecting examples to be used as training instances for a system which learns from examples. I discuss types of examples (e.g., near and solid misses), different mixes of examples, organization of a knowledge base of examples, and the role of various sources of knowledge (e.g., the system and its teacher) to be used in example selection. These ideas are illustrated with data collected in a series of experiments run by Buchanan, Rissland, et al. [1987].

1. INTRODUCTION

In most research on example-based learning techniques, like the candidate elimination algorithm and example-based generalization, the emphasis has been on the learning algorithm and not the examples used in applying it. That is, in the two-space view of learning, almost all the discussion has centered on the rule-space and not the examples-space. While this research is unquestionably important it tends to de-emphasize the other, I would argue, equally important, aspect of learning: examples and how to select them to give to the system and how they should be found or generated. This is the problem of *Intelligent Example Selection* or *IES*.

For human learning, it is no secret that the examples [2] one gives a learner greatly influence the content, style, and "efficiency" of the learning. A set of "bad" examples can even mislead the learner, especially where part of the learner's task involves induction and theory formation. In the context of hypothesis testing, a well-chosen example has as much power as a theorem, in that it can dispose of a conjecture by refuting it and thus completely re-direct the learning effort by pruning off unfruitful lines of investigation. These remarks are true in scientific and mathematical domains – as brilliantly discussed by Kuhn, Lakatos, and Polya – as well as in "softer" domains like law and philosophy – see Levi [1949] for an exposition of the situation in law, also [Rissland, 1984]. Well-designed ***what if*** and ***for instances*** can aid the knowledge acquisition process.

---

[1] This work supported in part by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval research under contract no. N00014-84-K-0017.

[2] *N.B.*, I use the word ***examples*** to cover both positive and negative examples of a concept.

These remarks are equally applicable to machine learning and discovery. For instance, Winston's work showed the efficacy of the near miss, Lenat's work showed how examples can direct the focus of learning (e.g., through example-dependent interestingness heuristics that prioritize the agenda) and Mitchell's work showed how examples cause changes in the version space (e.g., a massive cutting down, a small or no change, or even an over-shooting of S and G sets). However, there has been very little systematic study of the role of examples in the learning, except for Subramanian's and Feigenbaum's recent [1986] analysis, which addresses the problem of credit assignment and the work of Valiant on theoretical issues.

This paper will explore the following research issues on Intelligent Example Selection:

1. What types of examples are there? For instance, with respect to a "target theory" to be learned, one can speak of *positive* and *negative* examples, and *near* and *solid* misses as subtypes of negative examples. With respect to an expert's understanding, there are examples such as *start-up*, *reference*, and *counter*.

2. What mix of types of examples is most efficacious? For instance, what blend of positive examples and near and solid misses leads the learning system to perform best (e.g., for a performance task like classification, as measured by statistics like true/false positives and true/false negatives). And how should the mix change over the course of learning? For instance, are positive examples more "useful" in early stages (when broad inductive leaps are being made) and near misses in later stages (when the theory is being fine tuned)?

3. What information should the teacher use in selecting the examples? For instance, should it just use its *a priori* knowledge of the target theory (e.g., examples classified as positive, negative, near miss, etc., or according to some other scheme) or should it also pay attention to the "evolving theory" of the learning system. How much should the teacher pay attention to the learner's evolving theories?

4. What heuristics should the teacher use in selecting the examples? For instance, what is the efficacy of well-known Polya-heuristics like "Pick an extreme case". What knowledge about the target theory, in particular, about its corpus of examples and their interrelations – that is, its "Examples-space" – is necessary to implement such heuristics.

Clearly, definitive answers to such questions depend on factors concerning the learning system, its algorithm, its task, its context, etc. For instance, in one set of empirical experiments (the "RL experiments") [Buchanan, Rissland, Rosenbloom, Ng & Sullivan, 1987], the focus was on a top-down rule-learning system, called *RL* for Rule-Learner [Fu & Buchanan, 1985], which performed a classification task. As Winston pointed out in his seminal paper in 1975, IES issues need to be explored. They need to be explored across classes of learning systems and in the absence of theoretical analyses, part of this exploration needs to be empirical. One long range goal of research on IES is development of the understanding requisite for building an IES module which could custom

tailor example selection to particular knowledge acquisition problems on the basis of information such as the learning method, performance criteria, type of domain, difficulty of selecting/generating examples, etc. Ultimately, such an IES module could then act as an intermediary between the learning system and the source of the examples (i.e., between the two spaces of the two-space model for learning), and could drive a module that provides the examples with desired properties, for instance, either by generating them or selecting from an Examples-Knowledge-Base [Rissland, 1981a, b].

Research on IES is related to other lines of research: (1) understanding how human learners use examples to learn and to solve problems (e.g., in areas like mathematics, physics, law, electronic trouble-shooting, medical diagnosis); (2) how examples can be generated that meet certain desiderata (e.g., the constrained example generation model of [Rissland, 1980, 1981; Rissland & Soloway, 1981] and the "prototype-plus-deformation" model of [McCarty, 1980]); (3) how hypothetical examples are used and generated (e.g., in the law or strategic planning) [Rissland & Ashley, 1986; Rissland, 1987]; (4) the effectiveness of certain heuristics about examples (e.g., "Make a case weaker/stronger", "Use extreme cases", or "Enable a near-miss"); (5) example-based generalization, which, of the current machine learning methods, depends especially critically on selecting good examples; (6) indexing and memory issues as addressed by Case-Based Reasoning research (e.g., [Kolodner et al, 1985; Hammond, 1986; Rissland & Ashley, 1987]); and (7) research on connectionist-style systems (e.g., [Stanfill & Waltz, 1986; Lehnert, 1987]) which learn in response to a large corpus of training instances.

The import of work on IES for other work on machine learning is that with knowledge of how to intelligently select examples we should be able to improve knowledge acquisition, for instance, by either speeding it up [3] or improving its quality [4]. If we can understand what the essential components of IES are, we can design and implement better ("teaching") programs to drive learning systems and perhaps drastically reduce the need for human teachers to be ever-present in the learning cycle. This would contribute to breaking the knowledge acquisition bottleneck, for instance, in expert systems domains, like medicine or law, where there already exist large, well-codified, bases of examples, and a large part of training a learner (machine or person) is in selecting the training instances.

2. TYPES OF EXAMPLES

In this section, we describe some preliminary classifications of examples.

For starters, one should say what an **example** is. An example is an instance of a concept viewed from some perspective and for some purpose. Typically, examples are examples of something,

[3] That is, achieving a given level of performance with the use of fewer examples.

[4] That is, achieving a higher level of performance, with the use of a given number of examples.

like a concept or principle, and are used for some reason, like illustrating a principle or refuting a conjecture. As such an example tends to emphasize and encode only that information which is relevant in the given context to the item it is an example of. An example is an "understood" experience or set of data viewed through a certain lens and thus, can have a complex substructure [Rissland, 1978; Kolodner 1983a, b].

In the simplest computational realization, an example is simply a list of attribute-value pairs. That is, for each descriptive feature of a concept, or of the domain as a whole, there is a value, chosen from the allowable range for that feature. The descriptive features for a concept can come from a definition, for instance, an if-and-only-if rule. A set of domain features can come from some theory of the domain, for instance, a Meta-DENDRAL style "half-order theory" [Buchanan, et al., 1976].

The set of concepts one is trying to get the learning system to learn is called the **target theory** or **TT**. The version of the TT, that the learning system has learned, is called the **evolving theory** or **ET**. The simplist scenario is where the target theory consists of just one concept and that concept is defined by a single rule. In this case, there is no difference between learning a concept and learning a theory. [5]

In this discussion, for simplicity, the assumption is that concepts are defined by rules whose LHS is a conjunct of conditions and whose RHS is a concept name. These defining rules from the target theory provide necessary and sufficient conditions for membership in a concept class.

A variety of types of attributes can be used to describe concepts and examples (and thus occur as conjuncts on the LHS of defining rules). For instance, the features can be Boolean or can take values from a numerical range, an ordered set, or a symbolic hierarchy.

In this discussion for simplicity, an example will be taken to be a set of attribute-value pairs, where these range over the full set of attributes for the domain (and not just those mentioned in the concept's definition) and symbolic values are to be instantiated from leaf nodes of a value hierarchy tree. [6] This requires a feature to be instantiated even if not explicitly referenced in the concept's definition. This type of example could be called a "full, maximally specific" instance of a concept.

Alternatively, one could allow a descriptive feature not to have a value (i.e., not appear in the list of attribute-value pairs), or to have the value "do not know" or to be a non-leaf node from a value hierarchy tree. This would cause the Examples-space to be larger.

In the empirical experiments with RL [Buchanan, Rissland et al., 1987], the former – full, maximally specific – definition of an example was used. Winston used the alternative approach. The Examples-

[5] In general, I feel that learning a theory is much more complex. A theory is more than just the "sum" of its concepts; a theory also contains connections and relations between the concepts, among other things.

[6] For instance, consider a concept with the feature NAME whose value hierarchy tree had: as immediate descendants of the top (root) node *A* and *B*, names for abstract individuals, as leaf nodes *Aunt Tillie*, *Charlie*, *Whitman*, *Olivia*, names of specific individuals, and as intermediate nodes *Aunt*, *Family-Member*, etc. Then a maximally specific instance would have in its NAME feature, a value like *Whitman*.

space in the "full" approach is a subset of the Examples-spaces possible under the other approaches.

In summary in this discussion, an example is a full, maximally instantiated instance of a target theory concept defined by iff rules. The set of all examples of a theory X is denoted as $\mathbf{E}(X)$. The set of all positive examples of the theory is denoted by $E^+(X)$, and the negative examples is denoted by $E^-(X)$. See ***Figure 1*** for a suggestive depiction of the general situation for a theory with two concepts.

2.1 Positive and Negative Examples

A **positive** example of a concept is one that satisfies the conditions (i.e., the LHS) of at least one rule of the target theory having the concept as its RHS. Note it is possible for an example to be a positive example of more than one concept, if the concepts of the target theory are not disjoint. The set of all positive examples of a concept C is denoted by $E^+(\mathrm{C})$.

A **negative** example of a concept is one that does not satisfy the LHS of any rule in the target theory having the concept as its RHS. Note, it is possible for an individual example to be a negative example of more than one concept. A negative example that is a negative example of all concepts in the theory might be called an *anomalous* example since it is not "explained" by any of the concepts known in the theory. Note, the set of positive and anomalous examples is disjoint. The set of all negative examples of a concept, C, is denoted by: $E^-(\mathrm{C})$. See ***Figure 1*** for a suggestive depiction of the general situation for a theory with two concepts.

2.2 Random Examples

Since in our descriptions of mixes of examples, we need to describe examples having no particular attributes, we define the idea of a random example. Simply, a **random** example of a concept C is an example that has been selected randomly from $E(\mathrm{C})$, the set of all possible examples – in our case, from all the full, maximally specific instances – of the concept. A **positive random** or **negative random** example of a concept is selected, respectively, from $E^+(\mathrm{C})$ or $E^-(\mathrm{C})$. There are certain questions regarding sampling (e.g., with replacement or without replacement) which we put aside for this discusssion.

2.3 A Theory of Near Misses

Near misses comprise a class of examples of primary interest. We require a near miss to be a negative example of a concept from some theory – typically the target theory – such that it fails to be a positive example of the concept in just one aspect. This is similar to Winston's [1984, p. 393] definition, "A near miss is a sample that does not qualify as an instance of the class being

taught for some small number of reasons." It is surprising that even for what appears to be such a transparent idea, there are many subtleties which affect intelligent example selection.

We discuss three issues concerning the definition of near misses:

(1) metrics to assess "nearness";

(2) the referent for the near miss;

(3) "syntactic" vs."semantic" near miss.

2.3.1 Issue 1: How to Measure Nearness In defining near miss, one needs to know what metric (i.e., distance function) one is using. There are two obvious choices, which in some situations lead to different senses of nearness. Choice 1 is to define a measure of difference on the **descriptions** or the **representation** of the concept. Choice 2 is to define it **topologically** using the distance of an example to the set $E^+(C)$.

Choice 1 is what Winston used. He measured the differences representations of the target theory concept and the system's evolving model of it (e.g., by noting the presence/absence of certain pointers like SUPPORTED-BY). If there was one such difference, (or in some special cases more than one of the same type), this was called a near miss.

In the RL experiments, nearness was also measured using descriptions of a concept, in particular, the antecedents of rules defining it. If there was one difference between the description of an example and a rule's LHS, this was called near miss.

Choice 2 involves defining some distance function, a metric, on the space of examples, and thereby inducing a topology on it. In some situations, one can combine differences on individual "component" features in ways analogous to defining different metrics on a vector space: one can add the contribution from each feature or component (and thus get an l_1 or "city-block" metric) or combine them in a root mean square manner (and thus get the usual Euclidean l_2 metric) or one could take the max of the contributions (and thus get the *sup* or l_∞ metric). Each choice results in a different sense of nearness imposed on the space of examples.

The difference betweens these choices in senses of nearness is apparent if one has a situation as in *Figure 1*, where the sets $E^+(C_1)$ and $E^+(C_2)$ represent the sets of examples classified as positive using two different definitions, C_1 and C_2, of the *same* concept, C. One needs to say how to measure the distance of a particular negative example to the set, $E^+(C)$, which is the union of the two individual sets $E^+(C_1)$ and $E^+(C_2)$.

Sometimes, the topological approach seems clearly preferrable to the metric-on-descriptions approach. For instance, in the situation just decribed. A particular example can fail to satisfy one condition of one definition of the concept (e.g., a LHS of one rule) and yet still be a positive instance because it satisfies another. In this case, one has a near miss *a la Choice 1* because it fails a description but is actually not a miss at all. In this case, it seems one should attend to what the

concept means, as embodied in examples, and not how it is expressed. [7]

For the most part, we will stick with Choice 1 – metricize descriptions – even thought we sometimes appeal to the topological intuitions more in the spirit of Choice 2 – metricize the set of examples. There is no particular disadvantage to this in our discussion since we mostly confine ourselves to the simple scenario of the target theory usually having only one concept which is defined by only one rule. In more complex, and realistic situations, one would have to think more carefully about the choices and their ramifications.

2.3.2 Issue 2: What to Miss Off Of – The Choice of Referent This issue concerns the referent for the near miss. One can form a near miss of a concept with respect to different "levels" of reference, where the choice is in:

1. which theories to use,
2. given one or more theories, which rules to use,
3. given one or more rules, which conjunct to use.

Level 1: Which Theories to Use

The first level concerns **which** theory or theories to use in selecting the examples. This is especially relevant when there are one or more evolving theories, of the learning system's own manufacture to choose from, for instance, in an instructional mode where a teacher or example selector is tracking the learning system's evolving model of the target theory. Such choices might arise:

1. where there is a sequence of ET's, and the example selector can use the ET that: (a) is most recent; (b) is most highly refined; (c) covers the most true positive examples; (d) excludes the most true negative examples; (e) is that from several learning iterations back in the learning; etc.
2. where there is a conflict between ET's, because: (a) some correctly classified examples flip in their classification in successive ET's; (b) there are trade-offs in the statistics of true/false positives and negatives; etc.
3. where one ET is more "elegant" or otherwise desirable than another, for instance: (a) has fewer rules per concept; (b) has fewer conjuncts per rule; (c) uses fewer negations in the rules; etc.

Winston [1975], for example, uses both the target theory and the system's most recently evolved theory (i.e., its current model of the concept, like arch, being taught). A Winston-style near miss

[7] Of course, this type of problem would disappear if all rules were *iff* rules.

is an example that is negative with respect to the TT, negative with respect to the ET, such that there is only one difference between the representations of the near miss and the ET:

$$\text{For a Winston near miss } x\text{: } x \in E^{-}(TT) \cap E^{-}(ET)$$

Such a near miss is a negative example with respect to the TT and is classified as such by the system's ET. It causes a specialization of the ET, typically with the use of "must-be" or "must-not-be" satellite links.

For instance, in learning the concept of a two-block stack (Figure 5.xx in [Winston, 1975]), after having been presented with a two-block stack of block *A* on top of block *B*, the ET contains information that A and B are blocks and that one SUPPORTS the other. Now given a near miss of an unstacked stack, in which neither supports the other, the SUPPORTS link in the ET is revised to be a MUST-SUPPORTS link. Support thus has been learned to be a necessary condition. This near miss was negative modulo both ET and TT.

For some of Winston's near miss, it turns out that using a different approach as to what information is encoded in an example (recall Winston does not require a "full" example), the near miss will still be negative *module* TT but will change to being positive *module* the ET.

Consider the classic sequence involved in learning the concept of an arch (i.e., Figure 5.61 from [Winston, 1975] which is slightly altered in [Winston, 1984]), where the first example is the paradigmatic positive example of an arch and the second, a near miss with the lintel not on top of the other "supporting" blocks. At this point, the system's ET is pretty much the teacher's model of what an arch of toy blocks is: a brick block lintel which must be supported by two upright brick blocks (call them *A* and *B*). Note, that the fact that *A* and *B* do not touch is not recorded in either the positive examples or near miss presented thus far or in the ET. Now, given a near miss where the two supporting blocks do indeed touch, the program detects one difference: the presence of a TOUCH link in the near miss. In response, the ET is amended to contain a MUST-NOT-TOUCH link. Recounted this way, the near miss is again negative *modulo* both TT and ET.

Now, however, consider the case where "all" the information present in the examples and near misses is recorded – that is, consider ourselves in "fully specified" mode. Then the fact that *A* and *B* DO-NOT-TOUCH would be recorded in all the instances presented thus far and also the ET, and the near miss would now be **positive** *modulo* the ET and, of course, still negative *modulo* the TT.

Thus, differences in classification *modulo* the ET can arise from different assumptions as to the information present in an example. Note that there is also a change in the descriptors used to represent an example (e.g., NOT's and MUST's get introduced).

In some situations, by a change of referent theory, a Winston positive example can also be considered a near miss, even though Winston doesn't speak of it as such. Such an example, which would be incorrectly classified as negative according to the program's ET, is used by Winston to induce

generalization, for instance by redirecting an isa link up a value hierarchy.

For instance, continuing with the arch example. If one presents, as a positive example, an example which is the same as the initially presented paradigm except the lintel is now a wedge, one forces the program to generalize its ET on the basis of the single recognized difference between its current ET and this new example. The current ET incorrectly views the arch-with-wedge as a non-arch. The example is thus negative *modulo* the ET. Furthermore, it is in fact a near miss since there is but a single difference between it and the ET. Thus some of Winston's positive examples can also be viewed as near misses but with the referents "switched": that is, positive with respect to the TT and negative with respect to the ET:

$$\text{A Winston Positive Example } x\text{: } x \in E^+(TT) \cap E^-(ET)$$

In summary, Winstons' program does indeed derive great leverage from the near miss. Through change of referent, one can see near misses as the predominate type of example used.

In the RL-experiments, near misses were required to be negative *modulo* the TT, just as in Winston. Because RL was performing rule-refinement in a top-down manner, RL's revisions to rules caused LHS's to be more restrictive (e.g., through the addition of conjuncts and the addition of restrictions on the values allowable for already present conjuncts). Thus, the ET's were successive refinements of one another:

$$E^+(ET_{n+1}) \subseteq E^+(ET_n)$$

And, the target theory was a subset of all evolving theories:

$$E^+(TT) \subseteq E^+(ET_n)\ (\forall n)$$

This situation is depicted in *Figure 2a.*

Results from the RL-experiments indicate that using both the ET and the TT is better than using the TT alone, just as one might have expected. For instance, given 10 negative examples to use, choosing them from $E^+(ET) \cap E^-(TT)$ results in a statistics of approximately 90-95% correctly classified negatives ("TN's", true negatives) whereas choosing the examples from $E^-(TT)$ results in a statistic of 55% TN's [Buchanan et al, 1987]. [8] The RL experiments showed that the "best" examples come from the **annulus of effective examples** which are positive mod ET and negative mod TT:

$$x \in E^+(ET) \cap E^-(TT).$$

[8] A higher percentage of TN's is better since one is trying to drive false positives to zero, and (in percentages) TN + FP = 100%, where TN stands for true negative and FP for false positive.

See *Figure 2b.*

Furthermore, in the RL experiments, and in other top-down learning scenarios, because the ET's are shrinking in on the TT, the efficiency of the learning is increased by increasing shrinkage. This means to force the most shrinkage, the example selector should choose examples in the annulus as close to the border of the TT as possible. That is, the effective examples are examples that are positive mod ET and are near misses mod TT:

$$x \in E^+(ET) \cap NM(TT).$$

Other interesting situations arise (1) when the ET's are not a sequence of successive refinements or (2) ET and TT do not stand in subset relationship to one another but rather overlap partially. Such situations occur when there is a bifurcation in the space of possible ET's. (In such situations, Winston usually chose to proceed in a depth-first manner.) Notions such as elegance, coverage, or the ET's "distance" to the TT, could be used to frame different selection strategies of which ET to use and thus induce different styles of learning. To date, these questions have been largely unexplored.

In summary, there is a choice of which theories to use. There is the obvious matrix of the four possibilities involving ET and TT, in general, and these possibilities multiply when one considers multiple versions of the ET.

Level 2: Given a Theory, Which Concept Definition to Use.

If one uses all the rules defining a concept, one is missing with respect to the concept, or the theory of the concept, "as a whole". In this case, we are really considering the topological approach, discussed above. A near miss in this case can be perhaps best thought of in a set theoretic sense: it is outside the set of individuals which are positive instances of the concept, yet it is not too far outside; it is just over the boundary. As discussed above, to pursue this idea one would need define a metric to measure distance to the set of positive instances. The picture in *Figure 1*, where the theory has two overlapping concepts C_1 and C_2 is suggestive of the general situation.

For concepts defined by more than one rule, an individual rule could still be used as referent, but there are additional questions on how to select the rule. This would allow the possibility for a near miss with respect to one rule being a positive with respect to another since we are defining near misses in terms of LHS's of rules and not by where the example sits topologically with respect to $E^+(C)$. That is, using the LHS descriptive approach might lead to certain kinds of inconsistences, so that one must be very careful to state *modulo* which rule.

In the simplest of scenarios – a theory whose concepts are defined by a single rule – these difficulties don't arise. Note, even when the TT has only one rule per concept, the ET might not. This is the situation in RL. Thus, one might start out simply enough in the learning but be forced to confront problems of choice among rules on the ET "side", as one proceeds.

A near miss of a concept *with respect to a rule* then is a negative example of the concept that fails to meet exactly one precondition of the rule. The topological picture to have in mind here is the same as mentioned just above: a near miss example is just outside of the set of positive instances covered by the rule. See *Figure 2b*. Note that the difference between a near and "far" miss is lost for a concept defined by a rule with a single precondition: to miss a little is to miss a lot.

In summary, in this discussion a *near miss with respect to an individual rule* for a concept is an example that simply fails to satisfy exactly one of the preconditions of the rule.

Level 3: Given a Rule, Which Conjunct to Use

More refined definitions of near miss might require a miss on a particular condition or mix of conditions. These might come from knowledge of the relationships between the conjuncts (e.g., that even if there is no logical relation, two conjuncts may tend to be true or false at the same time) or from other knowledge like knowing which condition is the hardest to satisfy or most likely to fail or is most "important", etc. We won't pursue these.

In summary, there are several choices of referent for defining a near miss: choice of theories, choice of rule, choice of conjunct. Our near misses are made with respect to an individual rule, typically from the TT, and a miss on any single precondition will do.

2.3.3 Issue 2: How to Miss – Syntax vs. Semantics The issue of *how* to miss on an individual precondition focusses us on what is really yet another level of referent "below" that of the individual conjuncts. At issue here is how to choose the value to use in the near miss.

There are choices here also. A miss can simply miss the conjunct "as a whole" – by failing to match it – or it could miss more constrainedly by requiring a value "near" to the prescribed values. This distinction is not relevant for Boolean descriptors and it is only relevant for symbolic or numeric features where the range of possible values allows enough variation to distinguish between near and far values. For semantic near misses, not any value that fails will do.

To see this distinction, suppose we were working in a domain concerning heart disease and there were a descriptor for chloresterol, CHLOR, whose range is partitioned into three subranges: 0-200, 200-240, 240-350, called *low, mid*, and *high*. Suppose a conjunct in the defining rule for the concept of LOW-RISK is (CHLOR low) then a near miss **with respect to the conjunct as a whole** is simply an example with (NOT (CHLOR *low*)) – i.e., **any** value in the range not less than 200. A near miss **with respect to the conjunct's value** would require a value in the "nearby" *mid* subrange – i.e., from the subrange 200-240. The same idea carries through for symbolic values where one could require the value to be from a "nearby" node in the descriptor's value hierarchy tree, for instance, a sibling leaf node.

A near miss with respect to a conjunct "as a whole" will be called a **syntactic near miss**. A near miss with respect to the conjunct's value range will be called a **semantic near miss**. The choice here is whether or not to refer to the semantics defined by the values of domain features. In RL,

and other Meta-DENDRAL style systems, this semantic information is contained in the half-order theory.

Note, even taking into account such value range information, it is possible for there not to be any difference between a syntactic and semantic near miss. For instance, if the conjunct in the LOW-RISK concept in question were (CHLOR *mid*).

Syntactic Near Miss

Thus, a **syntactic near miss** for a given concept is an example that simply does not have the prescribed value for one of its preconditions. Any value that fails will do. No requirement is made that the value be near or far. See *Figures 3a, b.*

Note, since we require examples to be fully specified instances, the intuitive approach of simply dropping a conjunct from the rule and "positively" instantiating what remains reduces to the same thing as generating a syntactic near miss, since the example must have some value for the "dropped" feature. If we allowed "non-full" examples the situation might be different. The approach of dropping one conjunct at a time – which was essentially what Winston did – is one approach used by mathematicians to test the necessity of a condition. They assume the dropped condition is "silent", that is, not mentioned and not known about. Lawyers and judges also do this sort of necessity/sufficiency testing but they sometimes are quite careful with the difference between not knowing a value and knowing a wrong or different value. Condition-by-condition testing is one strategy to deal with the credit assignment problem [Subramanian & Feigenbaum, 1986] in situations where a problem can be decomposed into component conditions. In situations where the credit is not distributed "democratically" across components, this approach is not as effective. [9] How to deal with "silent" values has ramifications for issues in non-monotonic reasoning. Although we put aside consideration of such subtleties, we feel they are important and need to be addressed.

Semantic Near Miss

A semantic near miss for a given concept is an example that does not have the prescribed value for one of its preconditions **and** the value it does have is "near" to the prescribed value. **A nearby value** is: (i) for numerical features, is a neighboring value or from a neighboring subrange; (ii) for symbolic features, by a next nearest, usually sibling, leaf node as defined by the value hierarchy tree. In general when there is a partial or total order on the values one has a readily useable sense of near.

Figures 3 and *4* portray the situation for a concept defined in terms of two conjuncts with numerical ranges (i.e., simple intervals). For syntactic near misses, one gets "band aid" regions, with $E^+(C)$

[9] See the difference between "democratic" and "non-democratic" divide-and-conquer strategies [Rissland, 1977]. Examples of the former occur frequently in linear situations.

as the gauze and the near misses as the adhesive tape; see *Figures 3a, b*. For semantic near misses, one gets line segments, or "bars", just outside the rectangle for $E^+(C)$; see *Figures 4a, b*. The schematics suggest why semantic near misses are more effective than syntactic ones: they provide a much tighter bound on $E^+(C)$.

As we said, the force of the syntactic/semantic distinction only shows up in non-Boolean features: for instance, in numerical features with more than two subranges and in symbolic features with enough structure to their value hierarchy trees. One needs enough room to move around not only to be able to have a difference between near and non-near miss but also to be able to have a sense of semantic nearness.

2.4 Types of Non-near Misses: Solid and Far Misses

One can subdivide the set of negative examples into near misses and non-near misses. Subcategories of non-near misses are **solid misses** and **far misses**. Again, each of the three issues – nearness, referent, syntactic/semantic – is relevant here and can be viewed in the same way as for near misses. For simplicity, one can restrict oneself to the syntactic interpretation of missing.

A **solid miss** is a negative example that misses in more than one aspect – that is, with respect to more than one conjunct of a particular rule from a particular theory. The extreme case is the notion of a **far** miss, which is a negative example that misses in every respect.

2.5 Summary of Definitions for Negative Examples and How To Miss

In summary, negative examples of a concept can be subcategorized as:

- **near** – misses on exactly one conjunct;
- **solid** – misses on two or more conjuncts;
- **far** – misses on all conjuncts.

Each is defined with respect to a single rule for the concept from a theory – usually the TT. Each sort of miss can be done either syntactically or semantically.

The general scheme of how to miss can be summarized by the following nested decisions:

```
First select the theories to use.
Given a theory,
      select a rule.
      Given a rule,
```

```
select a conjunct.
Given a conjunct,
      alter it synctactically or semantically.
```

2.6 Extreme Examples

An **extreme example** is an instance whose values are extreme with respect to their ranges. An extreme example can be either positive or negative. Typically, one thinks of extremes only for concepts whose descriptors have some semantic richness, that is, some range of values allowing one to select extremes, like *max* or *min* values in the case of numerical features. In general, one requires some *ordering* information on the range of values (e.g., totally-ordered or partially-ordered). However, it is possible to think of an extreme example involving only Boolean conjuncts: here there's simply no difference between extreme and non-extreme. A more interested sense of extreme in the case of Booleans is when all the conjuncts are toggled the same way. What the interpretation of extreme value should be in the case of symbolic values with a value hierarchy tree, without some ordering on the leaf nodes, is not clear. If one did not require examples to be maximally specific, one could use notions of extreme generality and extreme specificity, which is useful in many domains, like mathematics [10] and law. [11]

In talking about extremes, there is the issue of whether to make the example *extreme in all respects* – that is, with respect to all conjuncts – or whether with respect to just one or some respects. Since we cannot *a priori* advocate one interpretation over another, we settle for being careful in our terminology (i.e., extreme **in-all-respects**, or **in-some-respects**, or **in-one-respect**).

Another aspect of extremes refers to the semantics of the values. In this sense, an **extreme** example of a concept with respect to a given feature is a example whose value is an extreme value of the range of values. A negative extreme for the concept C is taken from $E^-(C)$ and a positive extreme is taken from $E^+(C)$.

In the simple cases portrayed in *Figure 5*, the extreme examples ***in-all-respects*** occur at the corners of the rectangle for $E^+(C)$, inside it for positive extremes and outside it, for negative extremes. Extreme (positive) examples ***in-one-respect*** are the horizontal or vertical borders of the rectangle;

[10] In mathematics, one frequently defines a class of function (e.g., continuous) without addressing the class of its domain or range; then one examines the concept as one moves it through a range – hierarchy – of types like compact, closed, any set. These produce rules (i.e., theorems) which then contain an additional condition regarding the domain (e.g., A continuous function defined on a compact set is uniformly continuous, achieves its max and min, etc.) [Rissland, 1978].

[11] One often poses a sequence of hypotheticals where what is varied is the generality of some feature, like the name of the parties [Rissland, 1983]. For instance, there is a different gestalt between: (a) "*A* promised *B* $10,000." (b) "J.R. Ewing promised Mandy $10,000." and (c) "Aunt Tillie promised a nephew $10,000." Such variation can be used to great pedagogical or argumentative advantage.

the negatives of this type occur just outside these horizontal/vertical segments. The situations in *Figure 3* suggest why Polya's heuristic to "Try extremes" is so powerful in convex situations: they completely pin down $E^+(C)$. Of course, using conjuncts, without negatives, of features with Boolean, partially ordered, discrete, or numerical interval values to define concepts generates just such convex regions for $E^+(C)$. Allowing negatives of conjuncts on the LHS can decouple $E^+(C)$ into disjoint and therefore non-convex configurations. (This is one reason not to allow disjuncts in the LHS of rule definitions.)

Extremal Points

In mathematics, there is the notion (in certain contexts, like Euclidean spaces or locally convex topological spaces) of extremal points of a set: points which are not on any line segment between any two other points in the set. For numerical intervals, the extremal points are the end-points, the max and the min. For polygonal regions, the extremal points are the vertices (e.g., the "corners" of a square). In fact, there is a theorem – the Krein-Millman Theorem – which says that a non-empty, convex, compact set (in the right kind of topological space, like the R^2, which happens to be locally convex) is the closed, convex hull of its extremal points. (See [Roy, 1987] for a good survey.) That is, take the extremal points and generate all the line segments connecting them, and the closure of this set, and what you have is the whole set. Said another way, any point in a convex, compact set is either an extremal point or is an inner point on some line segment connecting two extremal points. These mathematical considerations suggest that by knowing the extremal points, we can fill in any of the information that goes between them.

Translating these ideas back to our example selection problem where we are trying to learn the boundary of the set of positive examples, $E^+(C)$, we see that with the right sort of value ranges, knowing about the extremes of $E^+(C)$, which we denote as $Ext(E^+(C))$, ought to completely define the situation.

2.7 Other Taxonomies of Examples

Effective indexing and management of the Examples-space is key to IES. There are many ways to index an example base, for instance, on frequency, typicality, failure; each allows quick access to examples of a particular known prior history. Where the examples are remembered in great detail, there is the possibility of applying approaches from research on analogy and case-based reasoning to problems in knowledge acquisition.

We have restricted our discussion so far to classes of examples that can be determined solely by examination of definitions of the theory (TT and/or ET) and its semantics (half-order theory). If we allow ourselves to reference additional information, specifically concerning relationships between entities (e.g., examples, rules) of the theory or an expert's understanding of it, we permit ourselves to use other taxonomies of examples.

To that end we recap one taxonomy concerning an expert's understanding [Rissland, 1978]. In that discussion, we were working in a tri-partite model of a theory: the major classes of entities were: (1) examples; (2) concepts and principles; and (3) results (e.g., rules, conjectures, theorems). Each group of entity formed a "space" of items organized in a semantic net with a link reflecting the primary mode of generating an entity: (1) in Examples-space, the link is "constructional derivation"; (2) in Concepts-space, the link is "conceptual dependency" of a definitional, and not the Schankian variety; and (3) in Results-space, the link is logical deduction. Other links like generalization/specialization are also possible, and are often implicity recorded. Examples are instances, and non-instances, of concepts and theorems (thus establishing inter-space links).

Based on this, we described the following taxonomy of examples:

start-up examples – examples which are readily understandable by a learner just beginning to work in a theory; effective start-ups are generalizable or are the precursors of reference or model examples;

reference examples – examples which are useful regardless of the learner's degree of sophistication or focus in the theory; references are "standard, textbook cases", good from all points of reference in the theory and at most stages of learning;

model examples – paradigms or generalized patterns suggestive of the general case;

counter-examples – examples used to refute or limit a concept or result;

anomalous examples – examples which don't seem to fit in with one's understanding and present some nagging conundrum which one feels needs to be resolved.

With regards to the taxonomies of concepts and results, a few classes are worth recalling:

key-results – results which are the major stepping stones in understanding the theory;

culminating-results – results which are the goal of the theory; they are the reason one is studying it in the first place.

With such a taxonomy, one could then frame heuristics like "Use start up examples for key results and important definitions in the early stages of learning", "Make sure eventually to use all standard reference examples", "Make sure eventually to use all standard counter-examples", etc. These are close to the heuristics used by experts – teachers and practitioners – in domains like mathematics.

3. Strategies for Selecting Training Examples

In general, a strategy for IES should specify the number, mix, and ordering of the examples to be selected by the IES. We assume the learning system can be presented batches of examples all at

once and that the order within the batch does not matter. At the end of each batch, the learning system has the opportunity to revise its ET. In the extreme case of a batch consisting of a single example, we are in the case covering Winston's program; where the batches contain more than one example, we are in the case that RL is in. Almost certainly, the order of the batches – that is, how one changes strategies over the course of a learning session – matters a great deal.

Thus, in summary, for simplicity, the strategies discussed here only specify the numbers and the mix of examples to be presented.

Note the strategies can serve two functions: (1) they can help define certain questions in empirical experiments; and (2) they can help define certain IES techniques in practice.

3.1 A Word About Sets of Near Misses

By a *random set of near misses for a concept* we mean that the conjuncts chosen for "missing off of" are selected at random. This means that it is possible for a conjunct to be chosen repeatedly or for a conjunct to be ignored entirely. This scatter shot approach amounts to sampling with replacement from the set of near misses. This applies to either syntactic or semantic near misses.

By a *systematic set of near misses for a concept* we mean that the set is formed by iteratively cycling over all the features mentioned in the LHS of a rule for the concept and over all the value subranges for the mentioned features. This amounts to sampling without replacement from the set of the near misses. Since it "loops" in some order through the features and their values, this sample is biased by the orders used. [12] Systematization biases the sample, although given a larger enough sample, one would hope the bias would wash out. Systematic sets raise the possibility of the set systematically overlooking certain features and values.

Of course, what we have described for approaches to the systematic-random issue are the extremes. There are hybrid possibilities: for instance, choose the features systematically and the values randomly, or vice versa.

3.2 Some Random Strategies

STRAT1: [**Ran(+, -)**] – This is the null strategy. It is the simplest strategy in the sense of requiring no knowledge of the ET or the TT for selecting the examples. [13]

[12] This could be done in either a "depth-first" manner – that is, an outer loop through the conjuncts and an inner loop through the values – or in a "breadth-first" manner – repeated loops through the conjuncts and one instantiation per loop. The choice is whether to exhaust the value possibilities for a given feature before moving on to the next feature.

[13] Once selected, knowledge of the TT is used to correctly classify the examples. In the case of a system learning without a teacher or oracle to classify examples, this rudimentary knowledge is absent. It is hard to see how much

STRAT1 uses a training set consisting totally of randomly selected examples of a given concept. Of course, within this random set there will be (most likely both) positive and negative examples; the actual proportions present depend on the random set used. Note it is possible for a random example to be another more interesting species of example – like a near miss – and that is why we define other strategies with a finer classification, and, therefore, control of examples.

In STRAT1 one has no controls on the mix of examples, only on their number. The main use of this strategy is its use to establish a baseline of performance against which to measure other strategies. It is also useful in situations where one does not want to expend the computational energy required to do anything more sophisticated – perhaps because the $E(TT)$ is not well-organized for more sophisticated selection – and that such a null strategy won't hurt performance – perhaps because the learning algorithm is insensitive to the overall statistical distribution of the $E(TT)$ (as is in the work of Valiant).

STRAT2: [Ran(+), Ran(-)] – This is a refinement of STRAT1 with the additional constraint that the random positives and negatives be chosen separately, in specified proportions, not necessarily equal. STRAT2 highlights the relative effects of negative and positive examples. In more refined strategies, one can then isolate the effect of specific types of negative examples (e.g., near misses, solid misses) by substituting them for the random negatives. STRAT2 is used for reasons similar to STRAT1, such as establishing a baseline by which to test hypotheses about the goodness of other IES strategies.

3.3 Strategies Involving Classes of Negative Examples

In these strategies, we substitute particular classes of negative examples for the random negatives used in STRAT2. The purpose here is to isolate the effectiveness of these particular classes. If one shows that a strategy, like STRAT3 which uses near misses instead of random negatives, does better – in some defined sense of better like minimization of false positives which is equivalent to maximization of true negatives – than STRAT2, one has established the efficacy of using near misses.

STRAT3: [Ran(+), NM(−)] – In STRAT3 near misses provide the only negative examples. The NM's are generated *modulo* some theory. No other type of negatives, like solid misses, are used in STRAT3. Note that the random positives and near misses are not necessarily given in equal proportion.

There are three points to note about STRAT3:

1. The proportions of random positives and near misses can be varied to determine the leverage provided by the near misses. Thus, one might them be able to discover rules of thumb such as

---

learning can happen in this case.

x near misses are "worth" y random negatives, where *worth* means that the system achieves the same given level of performance.

One can also run experiments to compare the performance of the system under STRAT2 and STRAT3 where one keeps constant the number of positive randoms and keeps equal the number negative randoms and near misses. By so doing, one can establish that near misses are "better" than random negatives.

In certain of the RL experiments, we varied the numbers to show the leverage of near misses and to validate the hypothesis that near misses (of any variety) are better than random negatives.

2. There are choices of how to generate the near misses and assemble sets of them: one can have random or systematic sets of semantic or syntactic near misses. This creates four sub-varieties of this strategy. [14] From the RL experiments, we concluded that (in the RL context), the difference in random versus systematic assemblage of the near misses did not matter much, but that the use of semantic or syntactic near misses did. Thus, here we note two particularly important sub-strategies:

   (a) **STRAT3a[Ran(+), NM-syn(-)]**
   (b) **STRAT3b[Ran(+), NM-sem(-)]**

   These strategies can be used to establish two facts: (1) By keeping all things constant and varying semantic or syntactic, that semantic near misses are better than syntactic near misses, and (2) By varying the numbers used, that semantic near misses are worth more, and discovering what number of each species is needed to establish a given performance level.

3. Of course, there is the question of what referent to use. In the RL experiments we used two: (1) TT alone; and (2) TT and ET together, where the near misses were taken from the annulus $E^{-}(TT) \cap E^{+}(ET)$ as described earlier.

STRAT4: [Ran(+), SM(-)] – Strategy 4 is completely analogous to STRAT3 with the use of solid instead of near misses. This strategy has all the subtleties and substrategies that STRAT3 does and can be used to establish the relative worth of solid misses vis-a-vis random examples and near misses and whether or not solid misses are better than other types of negatives.

The results of the RL experiments showed that solid misses (*modulo* TT and generated syntactically) were not worth as much as near misses, of either variety. Also, that a strategy using random negatives can be more effective than STRAT4 because it allows the possibility of including a near miss or two. Thus, at first blush solid misses seem hardly worth the trouble. However, in other contexts – for systems other than RL and for TT's more complex than those of the RL experiments

[14] Random positives, Ran(+), with a choice of syntactic near misses, generated randomly; semantic near misses generated randomly; syntactic near misses generated systematically; and semantic near misses generated systematically.

– things might be different. Basically, the trouble with solid misses in a top-down learning situation where TT $\subseteq$ ET is that they do not cause much shrinkage in the ET and thus not much increase in performance. In this sense, they might even be a "bad" type of example to use, since they affirmatively keep the learning system from shrinking its ET.

STRAT5: [**Ran(+), SM(-), NM(-)**] – This mixed strategy combines random positive examples and both kinds of misses – near and solid – as negative examples of a concept.

This strategy should have hybrid effects and is not particularly useful in experimentation to establish the efficacy of various example types. However, in practice, it might very well be useful because it forces the learning system to deal with a variety of exemplars. Questions of exactly what proportional mix, which varieties of misses, and which referents to use are most likely very dependent on the learning system and its current context (e.g., TT and current ET).

3.4 Strategies with Extreme Examples

There are many ways to use extremes, for instance, alone or in concert with randoms and specific subclasses of negatives, like near misses. To date, no experiments have been performed using these strategies, so we only point out two possibilities of the many we can envision.

STRAT6: [**Ext**($E^+(TT)$), **Ext**($E^-(TT)$)] – This could serve as a baseline. It captures the corners and edges of a situation such as that in *Figure 5.*

STRAT7: [**Ext**($E^+(TT)$), **NM-sem(ET)**] – This strategy should completely "box in" the border between TT and ET.

4. Discussion of the RL Experiments

Here we recap some of the results from the RL Experiments for convenience since they are not yet widely available.

The RL experiments explored and tested four hypotheses about IES:

> "H1: IES using ET and/or TT is better than random selection.
> H2: Near misses are better than solid misses.
> H3: Semantic near misses are better than syntactic near misses.
> H4: Using knowledge of TT and ET is better than using TT alone.
>
> [Buchanan et al, 1987]

In the data summarized here, unless other said: (1) random examples are chosen with respect to the TT; (2) near misses are chosen with respect to the TT. A sampling of the data supporting these hypotheses was:

1. With regard to H1 (Using TT and/or ET better than randomly choosing):
   - STRAT3a[20 Ran(+), 10 NM-syn] yielded performance of 70% TN's, whereas STRAT2[20 Ran (+), 10 Ran(−)] yielded performance of 35% TN's.
   - STRAT3a[20 Ran(+), 20 NM-syn] yielded performance of 85% TN's, whereas STRAT2[20 Ran(+), 20 Ran(−)] yielded a performance of 65% TN's.
2. With regard to H2 (NM's better than SM's):
   - STRAT3a[20 Ran(+), 10 NM-syn] yielded a performance of 70% TN's, whereas, STRAT4[20 Ran(+), 10 SM-syn] yielded a performance of 30% TN's.
   - STRAT3a[20 Ran(+), 20 NM-syn] yielded a performance of 85% TN's, whereas, STRAT4[20 Ran(+), 20 SM-syn] yielded a performance of 40% TN's.
3. With regard to H3 (NM-semantic better than NM-syntactic):
   - STRAT3a[20 Ran(+), 20 NM-syn] yielded a performance of 85% TN's., whereas STRAT3b[20 Ran(+), 20 NM-sem] yielded a performance of 90% TN's.
4. With regards to H4 (Using ET & TT is better than TT alone):
   - STRAT3[20 Ran(+), 10 NM-sem in effective annulus [15]] yielded a performance of 99% TN's.
   - STRAT3[20 Ran(+), 10 Ran(−)], where only TT was used, yielded a performance of 55% TN's.

5. Conclusions

In summary, this paper discussed the general problem of Intelligent Example Selection and laid out a conceptual framework for discussing the types and effect of examples in machine learning. In particular,

1. It developed a catalogue of various kinds of examples, including what might be called a theory of the near miss and extreme examples. It pointed out that there are three issues concerning the definition of near miss: (1) how to measure nearness; (2) what to measure nearness to; (3) what semantic information to use in defining nearness.
2. It catalogued an assortment of IES strategies specifying the numbers and mixes of examples to be used. These strategies provide: (1) a basis for empirical experimentation; (2) guidance for IES policies in practice.

[15] Recall the effective annulus is $E^{-}(TT) \cap E^{+}(ET)$.

3. We discussed the choices of knowledge to use in selecting the examples: (1) the TT alone; (2) the ET and TT; (3) the semantics of the domain.

4. We summarized results from the Buchanan-Rissland study to show the import of various of these concerns. That study showed that near-misses based on the semantics of the domain are more effective than ones based on purely syntactic considerations, that near misses of any variety are better than random examples, and that examples based on knowledge of the learning system's evolving theory of what it is trying to learn as well as the teacher's model of that theory is better than using the teacher's model alone.

5. We suggested how such IES considerations could improve knowledge acquisition, for instance, by enabling a system to achieve a certain level of performance with fewer "better" or more "effective" examples (like near misses) or by driving the system to higher performance with certain highly effective examples, like semantic near misses in what we called the "effective annulus".

Selected References

Buchanan, B. G., Rissland, E. L., Rosenbloom, P. R., Ng, H. T., & Sullivan, J. "The Role of Intelligent Instance Selection in Learning Systems: The Near Miss." Submitted for publication.

Buchanan, B. G., Smith, D. H., White, W. C., Gritter, R., Feigenbaum, E. A., Lederberg, J., & Djerasse, C. 1976. "Applications of Artificial Intelligence for Chemical Inference, XXII, Automatic rule-formation in mass spectrometry by means of the Meta-DENDRAL program. *J. Amer. Chem. Soc.* 98:6168-6178, 1976.

Fu, L., & Buchanan, B. G. 1985. "Learning Intermediate Concepts in Constructing a Hierarchical Knowledge Base". *Proceedings IJCAI-85*, pp. 659-666.

Hammond, K. 1986. "CHEF: A Model of Case-Based Planning. *Proceedings AAAI-86*, pp. 267-271.

Kolodner, J. L. 1983a. "Reconstructive Memory: A Computer Model." *Cognitive Science*, 7(4): 281-328.

Kolodner, J. L. 1983b. "Maintaining Organization in a Dynamic Long-Term Memory." *Cognitive Science*, 7(4): 243-280.

Kolodner, J. L., Simpson, R. L., & Sycara-Cyranski, K. 1985. "A Process Model of Case-Based Reasoning in Problem Solving. In *Proceedings IJCAI-85.*

Lehnert, W. L. 1987. "Case-Based Problem Solving with a Large Knowledge Base of Learned Cases", In *Proceedings AAAI-87.*

Levi, Edward H. 1949. *An Introduction to Legal Reasoning.* University of Chicago Press.

McCarty, M. T., & Sridharan, N. S. 1980. "The Representation of an Evolving System of Legal Concepts." In *Proceedings Third National Conference of the Canadian Society for Computational Studies of Intelligence*, Victoria, B.C., May 1980.

Rissland, E. L. 1984. "The Ubiquitous Dialectic". *Proceedings Sixth European Conference on Artificial Intelligence (ECAI-84)*, Pisa, Italy, September 1984. (Published by North-Holland).

Rissland, E. L. 1983. "Examples in Legal Reasoning: Legal Hypotheticals". *Proceedings IJCAI-83*, Karlsruhe, Germany.

Rissland, E. L., 1981. *Constrained Example Generation.* Technical Report 81-24, Univ. of Mass, 1981.

Rissland, E. L. 1980. "Example Generation". In *Proceedings Third National Conference of the Canadian Society for Computational Studies of Intelligence*, Victoria, B.C., May 1980.

Rissland, E.L. 1978. "Understanding Understanding Mathematics". *Cognitive Science*, Vol. 2, No. 4.

Rissland, E.L. 1977. "Epistemology, Representation, Understanding and Interactive Exploration of Mathematical Theories." Ph.D. Dissertation. Dept. of Mathematics, MIT.

Rissland, E. L., and K. D. Ashley. 1987. "HYPO: A Paradigm in Case-Based Reasoning". *COUNSELOR Project Memo*, 1987.

Rissland, E. L., and Ashley, K. D. 1986. "Hypotheticals as Heuristic Device." In *Proceedings AAAI-86*, pp. 289-297. Philadelphia, August 1986.

Rissland, E. L., and E. M. Soloway. 1981. "Constrained Example Generation: A Testbed for Studying Issues in Learning". *Proceedings IJCAI-81*, Vancouver, B.C., 1981.

Roy, N. M. 1987. "Extremal Points of Convex Sets in Infinite Dimensional Spaces" *The American Mathematical Monthly*, Vol. 4, No. 5, pp. 409-422, May, 1987.

Stanfill, C., & Waltz, D. 1986. "Toward Memory-Based Reasoning", *CACM*, Vol. 29, No. 12, December 1986, pp. 1213-1228.

Subramanian, D. & Feigenbaum, J. 1986. "Factorization in Experiment Generation" *AAAI-86 Proceedings*, pp. 518-522.

Winston, P.H. 1975. "Learning Structural Descriptions from Examples," in *The Psychology of Computer Vision*, edited by Patrick H. Winston, McGraw-Hill Book Company, New York, NY.

Winston, P.H. 1984. *Artificial Intelligence*, 2nd Edition, edited by Patrick H. Winston, Addison-Wesley Pub. Company, Reading, MA.

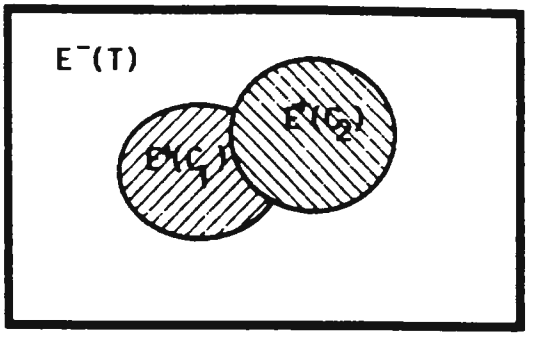


Figure 1 - A theory, T, consisting of 2 concepts C_1 and C_2. $E^+(T) = E^+(C_1) \cup E^+(C_2)$.

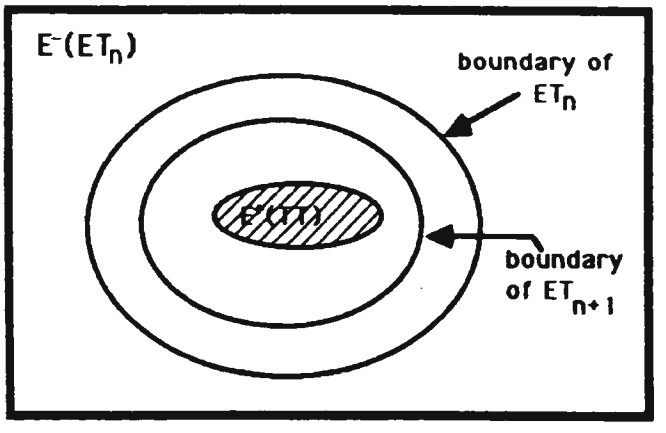


Figure 2a. The shrinking of evolving theories, all of which contain the target theory.

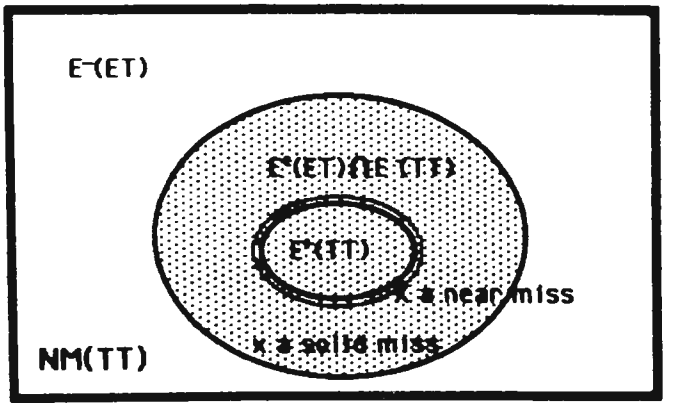
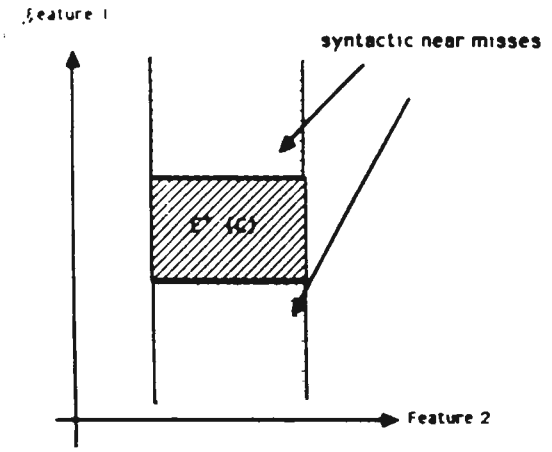
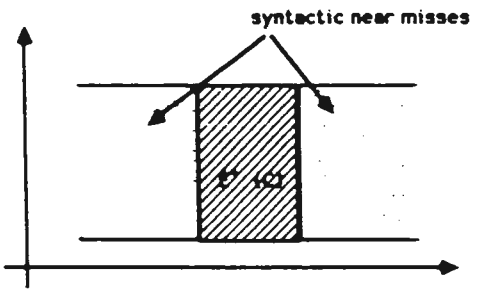
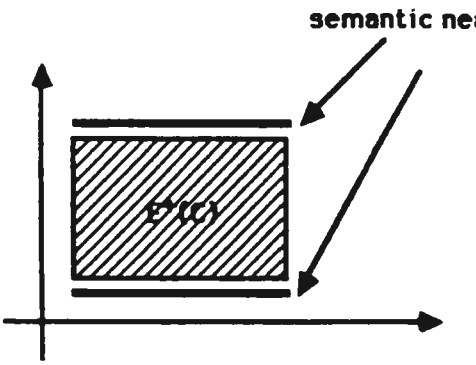


Figure 2b - The annulus of effective examples.

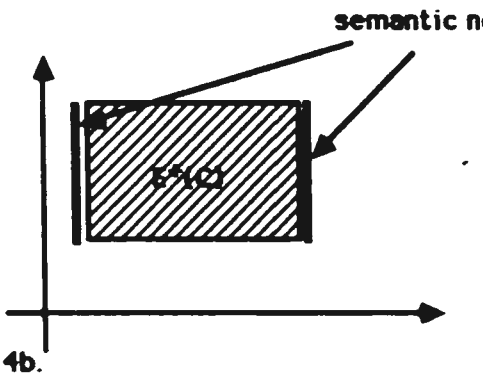




gures 3a and 3b - The syntactic "band-aids' of near misses for a concept defined in terms of two numerically-valued features.



4a.



4b.

Figure 4 - Semantic near misses for a concept defined as the conjunct of two numerically-valued features reside in the bars just outside the intervals of acceptable values.

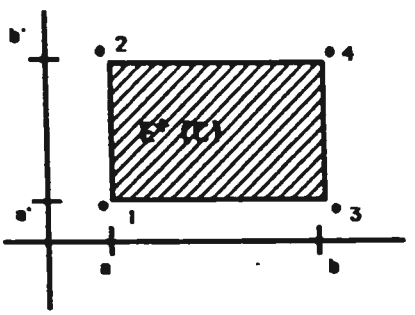


Figure 5 - Extremal points for a concept C.
The four corners of $E^+(C)$ are extreme in-all-aspects positive examples.
Points 1,2,3,4 are (minimum) extremes for $E^-(C)$.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

A Formal Analysis of Machine Learning Systems for Knowledge Acquisition[1]

Valerie L. Shalin
Edward J. Wisniewski
Keith R. Levi
Paul D. Scott[2]

Honeywell
Systems and Research Center
3660 Technology Drive
Minneapolis, MN 55418

Abstract

Machine learning techniques can be of great value for automating certain aspects of knowledge acquisition. Given the potential of machine learning for knowledge acquisition, we have begun a systemtic invesitgation of how one might map the functions of knowledge-based systems onto those machine learning systems that provide the required knowledge. The goal of our current research is to provide a general characterization of machine learning systems and their respective application domains.

INTRODUCTION

Traditionally, the design of knowledge-based, expert systems has involved a process of interaction between a domain expert and a knowledge engineer. During this interaction, called *knowledge acquisition*, the knowledge engineer formalizes the expert's knowledge as inference rules and data structures. There are a number of obvious problems with this approach. First, developing an expert system can require an enormous amount of effort. A system may fail to perform properly in its application domain and require additional modification. Typically, this will involve continuing, ill-defined interactions between expert and engineer during which the system is gradually modified as performance failures arise. Second, although an expert may exist for a given domain there is no guarantee that the expert can adequately access and describe his or her own knowledge so that can easily be represented in a program. Finally, as the first problem implies, it is difficult to determine when the knowledge base is correct, consistent, and complete. Knowledge acquisition techniques amount to a process of generate, test, and modify. A system is considered reliable when it performs well for some arbitrary period of time.

In view of the problems with current techniques, researchers have been interested in automating the process of knowledge acquisition. We see at least four ways that machine learning (ML) might facilitate knowledge acquisition for the knowledge requirements of

---

[1] This research was supported by funds from AFWAL, Contract No. F33615-86-C-1125.

[2] Department of Electrical Engineering and Computer Science, University of Michigan

expert systems. First, machine learning systems might contribute to the *initial* construction of a knowledge base. Several machine learning systems have been used to derive an initial set of rules for knowledge-based systems (AQ11, see Michalski and Chilausky, 1980; ISG, see Wisniewski, Winston, Smith, and Kleyn, in press). Second, machine learning systems might refine *existing* knowledge bases. This idea spawned the development of smart rule editors (e.g., TEIRESIAS, see Davis, 1979) and learning apprentice systems (e.g., LEAP, see Mitchell, Mahadevan, and Steinberg, 1985; Smith, Winston, Mitchell, and Buchanan, 1985). Third, machine learning systems might be helpful in *adapting* a knowledge-based system, for example to accommodate user expertise or style. Finally, in general, machine learning may provide a principled method for constructing knowledge bases of expert systems. Machine learning may replace the ill-defined knowledge engineer-expert interaction. Thus, it provides a *formalized mechanism* for the knowledge acquisition process.

Given the potential of machine learning for knowledge acquisition, we have begun a systematic investigation of how one might *map the functions of knowledge-based systems onto those machine learning systems that may provide the required knowledge.* The goal of our current research is to *provide a general characterization of machine learning systems and their respective application domains.* Automated knowledge acquisition may be feasible to the extent that these application domains match in an abstract (theoretical) manner, the application domains of expert systems. This requires an analysis of both machine learning and application domains in terms of common issues and at a proper grain size. We believe that the payoff of this investigation extends beyond the practical problem of knowledge acquisition by yielding a better general understanding of the nature of knowledge in knowledge based systems.

A crucial consideration for a successful match is the abstraction of an appropriate "learning problem" from an application domain. Specifically, a machine learning system will rarely provide all of the knowledge for a complete expert system function. Rather, we must select a piece of the problem that will be handled by the learning system within the context of a larger performance system. Our approach to identifying relevant machine learning systems for knowledge acquisition involves four steps:

- First, we characterize machine learning systems according to a general framework so that we can evaluate systems comparatively according to a standard set of issues.
- Second, we characterize the general learning problems in an expert system that can be addressed by machine learning systems.
- Third, we assess the match between a machine learning system and a learning problem according to a set of dependencies that we have identified, between features of machine learning systems and those of application domains.
- Finally, we evaluate a learning problem/machine learning system match based on the satisfaction of these dependencies.

In the remaining sections, we present this approach in more detail. In Section 2, we describe a general framework for analyzing machine learning systems. In section 3 we describe our approach to identifying the learning problems in expert systems that may be addressed by machine learning. In section 4 we define and illustrate the idea of dependencies. In section 5 we apply the approach established in sections 2, 3, and 4, to recommend machine learning systems for three different knowledge acquisition problems. In section 6 we describe the future work required to complete the approach.

A GENERAL FRAMEWORK FOR ANALYZING MACHINE LEARNING SYSTEMS

In this section we first describe a general framework for analyzing the components of machine learning systems. Our framework is based on one of the author's analysis of machine learning components (Scott, 1987). This framework explicitly identifies components found in all machine learning systems that are often only implicitly identified by their developers. We then describe the role of a teacher in machine learning systems according to the different machine learning components. As we shall describe, the role of a teacher greatly affects the practicality of machine learning applied to the knowledge acquisition problem for expert systems.

Machine Learning System Components. Consistent with other researchers (e.g., Michalski, 1986; Scott, 1983, 1987), we define learning as any change in a system's representation of its environment. In particular, consider a system that uses a specific representation scheme, called RS. The power set of this representation scheme, R*, denotes all the possible representations that the system can build, using RS. Specifically, Scott (1987) views learning as the task of searching R* for an acceptable representation of the application domain. During this search, the system is subjected to a sequence of experiences from the application domain, E_1, E_2, E_j, E_f. Scott calls this sequence the *experience trajectory* . If R_i denotes a subset of R*, the behavior of any learning program can be characterized as the sequence, R_0, R_1, ...R_j ...R_f where R_j is the set of representations being considered by the system after the jth experience, E_j, R_0 and R_f are the initial and final representations, respectively. Scott calls this sequence a *learning trajectory* .

Scott views a learning program as a function that generates the learning trajectory. Specifically, a learning procedure, L, has the form: L: R* X E ——> R*, where E is the set of all possible experiences. This function depends upon the existence of the experience trajectory, which itself can be described by a function: G: R* X W ——> E. Here, W denotes the application domain or world in which the system is learning.

The learning procedure requires a function that evaluates its representation. Typically, this function informs the learning algorithm that it has found a satisfactory representation. It may also guide the search toward such a representation. We describe the evaluation function as V: R X E ——> Q, where R is the set of representations and Q the set of possible evaluations. In the simplest case, V maps representations onto two values--- satisfactory and unsatisfactory.

Given this terminology, we define a *learning system* as a 3-tuple, {L, G, V}, where L generates new sets of representations, G generates new experiences of the application domain, and V evaluates the representations. A learning system operates in a *learning environment* . This is a 4-tuple, {W, RS, E, Q}, where W is an application domain, RS a representation scheme, E a set of possible experiences, and Q a set of values of representations. A learning system interacts with its environment, searching through a set of representations that describe the application domain.

Role of External Agent in Machine Learning System Components. An *external agent* may *assist* a learning system in this search through its representation space. In particular, an external agent can provide some combination of the system's representation generator (L), experience generator (G), and representation evaluator (V). Therefore, we can categorize learning systems into eight distinct groups. At one extreme are systems that take full responsibility for the generation of their experiences, and the

generation and evaluation of their representations. At the other extreme, an external agent provides all these functions to the system. In Table 1, we have grouped representative machine learning systems into these categories.

This general framework classifies machine learning systems according to an important criterion for knowledge acquisition---the role of external guidance in the learning process. Systems requiring human intervention may succeed or fail in a domain to the extent that this intervention can be provided and to the extent that we fully understand the function of this intervention. A machine learning program that requires specialized input from an external agent may not be useful for on-line knowledge acquisition in real-time systems.

Let us illustrate the grouping of systems according to the role of an external agent with examples of systems that fall toward the extremes. TEIRESIAS is a program that learns new inference rules via user interaction (Davis, 1979). It can assist an expert system in knowledge acquisition. The user is the experience generator, providing problems for TEIRESIAS to solve. The user and to some extent, the system, function as the representation generator. Here, the user provides an initial representation (in the form of a rule set). During routine problem solving, the system automatically integrates information provided by the user into new or modified rules. Finally, the user primarily evaluates the representation, informing the system when it has failed to apply a rule or has incorrectly applied a rule. Although TEIRESIAS performs some representation evaluation via its rule models in general, an external agent provides the learning components of TEIRESIAS.

At the other extreme, consider the program LEX, which acquires problem solving heuristics in the domain of of symbolic integration (Mitchell, Utgoff, & Banerji, 1982). LEX contains a *problem generator* that is responsible for the system's experiences. Using a credit assignment module (called the *Critic*) and a generalization algorithm, LEX proposes and refines heuristics to improve problem solving performance. These components function as the representation generator and evaluator. Therefore, LEX does not rely on external guidance during learning. Such a system is independent of any external guidance. Therefore, it might function well as an on-line learning component of an expert system.

IDENTIFYING LEARNING PROBLEMS IN EXPERT SYSTEMS

We view a machine learning system as potentially fulfilling a knowledge requirement for an expert system application domain. According to the framework that we just presented, the expert system takes on the role of the domain (W) in a machine learning system. With this view in mind, the next step is to provide an analysis of the knowledge required for an expert system.

In this section we describe a working set of these knowledge requirements. We focus on their content (rather than their representation in a particular expert system architecture), similar to idea of knowledge level concepts that are emerging in the AI literature (Newell, 1982; Dietterich, 1986). We have assembled this set of knowledge requirements from three main sources. We have made use of existing concepts in the machine learning literature, such as knowledge of category names, knowledge of input features, knowledge of rule applicability conditions, etc. We have also made use of relevant discussions in problem solving theory (Greeno & Simon, 1986) and expert systems (Clancey, 1983; Hayes-Roth, Waterman and Lenat, 1983) on the nature of knowledge for expert problem solving. To structure this set of knowledge requirements, we have grouped them as they pertain to five general expert system functions (see Table 2). Our expert system categories are based on existing taxonomies of expert systems, (Hayes-Roth, et. al., 1983), problem solving (Greeno et. al., 1986), and a report on expert systems for pilot-aiding (Schudy, Bobrow, Anderson and Wilcox, 1986) although we note that the boundaries between

Table 1
A Principled Taxonomy

Role Of Teacher	Common Description	System or Theory
(L, G, V)	Interactive Knowledge Acquisition	TEIRESIAS
	Learning by Being Told	FOO
(L, V)	Certain Aspects of Interactive KA	LEAP
(L, G)	Student Modeling	LMS
(G, V)	Concept Learning	ID3
		Version Spaces
		AQ11
		Winston's ARCH program
		Genetic Algorithms
(G)	Clustering	Cluster2
		BACON
		Competitive Learning Theory
		BSB
	Theory Driven Learning by Example	EBG
		EBL
	Planning and Problem Solving	SOAR
		STRIPS
		Samuel's Checkers Playing Program
	Student Modeling	ACM
(V)	Reinforcement	Classifier Systems
		Boltzmann Machines
(L)		
( )	Learning by Discovery	EURISKO
		LEX

Table 2. General Expert System Functions and their Knowledge Requirements[a]

Classification (and Monitoring)	Interpretation
Concepts for Input Features	*Rules of inference
*Class concepts based on feature overlap[b]	Strategic knowledge
*Class concepts based on functions context	Cost of additional data
*Higher order features	Stop conditions
*Correlations between features	
*Probability distribution of exemplars	
*Hierarchical class structure	

Design	Problem Solving and Planning
Constraints on objects	Executable operators
Test for solution states	*Applicability conditions of operators
	*Macro functions
	Domain goals and subgoals
	Planning operators
	Constraints between executable operators
	Planning heuristics

[a] Knowledge requirements for more complex functions subsume requirements for less complex functions.

[b] Starred knowledge requirements can be fulfilled by ML.

functions are somewhat fuzzy. The knowledge requirements for more complex expert system functions subsume requirements for less complex expert system functions. In the remainder of this section we describe the knowledge requirements associated with each expert system function. We then indicate candidate machine learning systems that are capable of supplying them for expert system knowledge acquisition.

We define *classification* to mean recognition. A *classification* function matches the input features of an exemplar of a class to a concept or *internal* representation of the class. Knowledge requirements include concepts for the input features and concepts for the classes. Input features may be static and context independent, in which case the required concepts may be based on rules that maximize within category similarity and between category dissimilarity. Input features may relate to functions of the exemplars. In this case the required concepts may be based on functions of the exemplars (e.g., the class of sitable things, the class of drinkable-from things). If the classification task is context sensitive, there is an additional knowledge requirement to represent the context. Other classification knowledge may include knowledge of important relations and higher order features (e.g., exclusive-or), knowledge about correlated features (weak constraints), knowledge about the probability distribution of exemplars, and knowledge about hierarchical class structure.

A *monitoring* function is similar to classification of input with the emphasis on detecting changes in input over time. Knowledge requirements include those of classification. Knowledge of what features to monitor relates to the consequences of the detected change. For example, monitoring may serve to detect and warn about nullified plan rationale or plan constraints thereby invoking a new plan. Monitored features are often numerical, with the idea of significant change based on exceeded thresholds or ranges.

An *interpretation* function constructs a coherent representation from classified objects. Examples of interpretation functions are scene analysis, prose comprenhension or medical diagnosis. Pre-existing interpretations may not exist, or if they do, they require a significant amount of sequential reasoning and search to locate. Interpretation knowledge requirements include those of classification. Interpretation also requires rules of inference (operators), and strategic knowledge for efficiently constructing a new interpretation using these rules, e.g., ordering screening clauses in medical diagnosis (Clancey, 1983). In order to build the interpretation efficiently, interpretation functions may also require knowledge about the cost of obtaining new data, or make use of correlated features or knowledge about the probability distribution of the possible interpretations. An important, but easily overlooked, requirement is for "stop conditions" that either implicitly declare the inputs to be "understood" or redirect the interpretation to resolve contradictions.

A *design* function arranges objects according to constraints on those objects. An example of a design function is to configure Vax components. Knowledge requirements include a representation of objects in terms of features that can be related to the desired constraints. These features must also eliminate unacceptable pathways in design space, hence the important strategic function of a proper representation for successfully solving design problems. Designing also requires tests for recognizing the solution states that meet design constraints.

Problem Solving and Planning arrange actions according to constraints on action sequences. We distinguish problem solving and planning from interpretation by their interaction with the world, and capacity to permanently change the internal and external environment. Problem solving concerns reasoning about states and state changes. Planning concerns problem solving about actions. Knowledge requirements include: operators expressed in the representation language of executable functions, knowledge of the applicability conditions for executive operators, effective sequences of operators (macro

operators), strategic knowledge, and knowledge of domain subgoals and goals and tests for confirming that a goal state has been achieved. Planning requires, in addition, planning operators and their applicability conditions, knowledge of constraints between executable operators or sequences of actions (perhaps at multiple levels of abstraction), and planning heuristics.

Having thus defined a set of possible knowledge requirements of expert systems, we now identity machine learning systems that can supply the required knowledge. In Table 3 we have idenfied the knowledge supplied by some of the standard machine learning systems. We have arranged the table so that knowledge requirements that pertain to classification appear in the top two-thirds of the table, and knowledge requirements that pertain to problem solving and planning appear in the bottom third of the table. On the basis of knowledge supplied, it is clear even with this limited sample that there are numerous candidate machine learning systems relevant to individual expert system knowledge acquisition problems. For example, there are eight systems that can supply concepts for classification based on feature overlap. In order to select the most relevant machine learning system, one must compare the systems in terms of their ability to address a given knowledge acquisition problem. We address this question in the following section, based on the idea of dependencies between machine learning systems and their appropriate learning problems.

DEFINING DEPENDENCIES BETWEEN MACHINE LEARNING SYSTEMS AND LEARNING PROBLEMS IN EXPERT SYSTEMS

We have used Scott's formalism as a basis for substituting an expert system application domain for one component of a machine learning system—its learning domain (W). However, as we discuss below, the learning domain affects the other components of a machine learning system (RS, E, G, L, V). In particular, there are dependencies between W and the other machine learning system components. In this context, a dependency specifies how a characteristic of the domain affects one of these components (or vice versa). To take a simple example, consider the problem of learning about a rare event. This characteristic of the domain suggests that there might be few learning experiences (E) available to the machine learning system. Further, it suggests that the representation generator (L) must be able to learn from a few examples. One could use these dependencies to partially justify why, for example, a theory-based learning system might be appropriate for this learning problem rather than a similarity-based system. In general, theory-based systems require fewer examples to learn than similarity-based systems.

To uncover these dependencies, we have developed a set of questions designed to characterize expert system application domains and machine learning systems in detail. Twenty of these questions correspond to the learning domain and form the basis of our domain template. Fifty questions correspond to the other machine learning components (RS, E, G, L, V and Q) and form the basis of our system template. As shown in Figure 1, the questions in the domain template have counterparts in the system template, reflecting dependencies that exist between the application domain (W) and other machine learning components. This establishes a mapping between machine learning systems and expert system applications domains. One can note those dependencies that are supported and violated by a particular machine learning system/application domain pairing. In this manner, one can identify relevant machine learning systems for knowledge acquisition in the application domain.

We based the domain template on the analyses that expert system developers have used as design guidelines for building expert systems (Kline and Dolins, 1985, Hayes-Roth, Walerman and Lenat, 1983). The domain questions allow us to identify properties of the

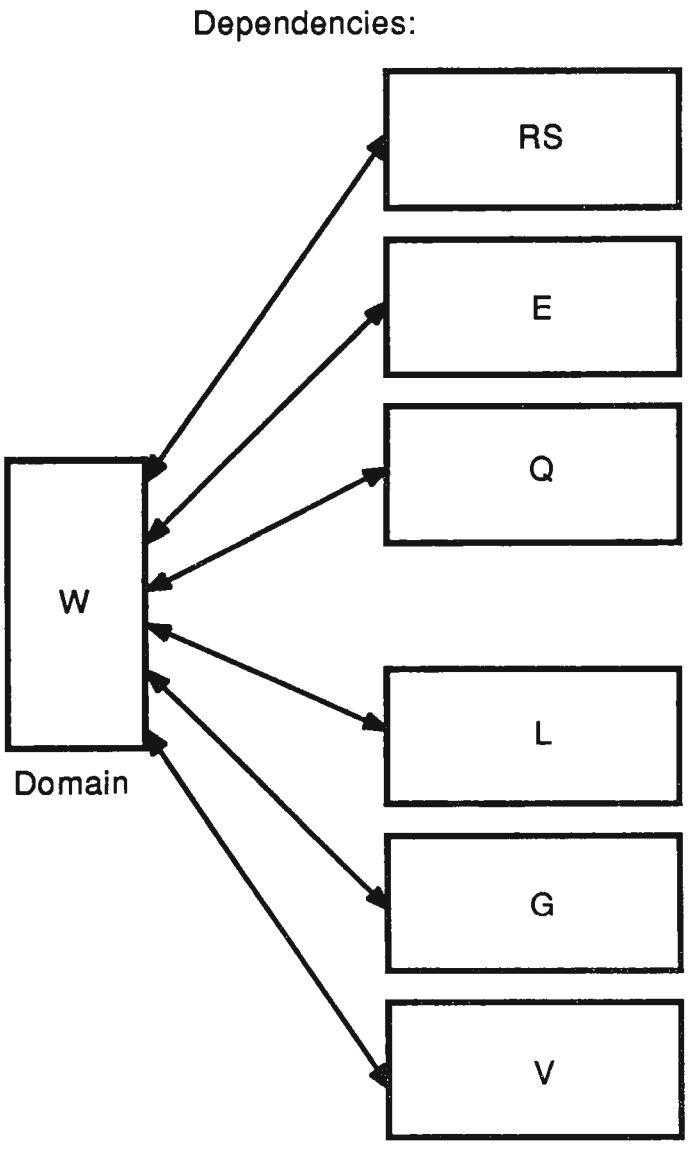


Figure 1. Dependencies between the learning domain and other ML system components.

KL6.1.3

	TEIRESIAS	FOO	WINSTON	VERSION SPACE	ID3	AQ11	CLT	ACM	STRIPS	SAMEULS	SOAR	EBG	BOLTZMANN	BSB	CLUSTER2	EBL	BACON	LEX	EURISKO
Concepts for Classes Based on Feature Overlap				√	√	√	√						√	√	√				
Concepts for Classes Based on Function			√									√				√			√
Higher order Features							√						√	√					
Correlations Between Features							√						√	√					
Probability Distribution of Exemplars													√						
Hierarchical Class Structure (e.g., decision tree)					√										√				
Empirical Relations Between Quantitative Variables																	√		
Rules of Inference	√																		
Applicability Conditions of Operators								√		√	√							√	
Macro Functions									√	√									
Constraints Between Executable Operators		√																	

Table 3. Knowledge Supplied by ML Systems

expert system in the application domain of interest, such as reasoning strategies, control structures, and representational formalisms. Table 4 illustrates some of the questions in the domain template. We developed the first example question because general types of knowledge imply different kinds of representational formalisms and reasoning strategies. For example, Davis and King (1977) wrote that some domains consist of many different independent states (e.g., medicine) while others can be characterized by unifying principles that emphasize the similarities of seemingly different states (e.g., well-established areas of physics). The former domains are best represented and reasoned about with rules and production systems, the latter with algorithms and mathematics. In general, the selected answer to this question has a number of implications for appropriate representational formalisms and the information that might be available to support a learning problem. These are too numerous to list here. Other questions in the domain template establish the size of the learning problem (example 2) and the nature of the learning experiences (example 3).

In our system template we identified the characteristic features of each of the learning components of a given machine learning system. We designed these questions to complement the domain template. For example, in (1) we asked about a given system's representation scheme, as illustrated in Table 4, to complement the adjacent question in the table from the domain template. In example 4, we asked about the system's capability to handle both positive and negative examples. In general, positive examples allow a system to generalize and negative examples allow a system to specialize. There may be a potential mismatch between the types of examples that a machine learning system strongly depends on and those that are provided by a domain. For example, EBG depends only on positive examples and does not interact with negative examples. It will not be appropriate for problem domains that do not tolerate overgeneralization (false alarms). Similar complementary questions occur in the system template for all of the domain questions. We have identified 90 such dependencies. There are more dependencies than questions because some questions pertain to multiple questions in the complementary template.

RESULTS

We have applied our system template to 19 machine learning systems that are representative of the field. We have analyzed twelve learning problems, selected from the domain of pilot aiding. In this section, we present two examples of the twelve learning problems we have analyzed. There are three important components of each example: 1) The target performance problems, 2) the abstracted learning problem, and 3) the reasons for selecting candidate machine learning systems. These three components are present in every machine learning application. However, in most discussions of machine learning systems the three components are never explicitly considered or reported. Rather, the selection of a target performance problem, an appropriate learning problem, and an appropriate learning system have typically been intuitive decisions based on years of hands-on experience and research with particular systems. This study represents a first attempt to systematically and explicitly address and report the important considerations for selecting machine learning systems for a target performance problem.

Although we report the three stages in a sequence, the process cannot be strictly serial in practice. Choosing an appropriate performance problem and then abstracting a viable learning problem requires insight into the capabilities and requirements of existing machine learning systems. On the other hand, the critical capabilities and requirements of various machine learning systems become apparent only after one has attempted to define a learning problem for a performance domain. Thus, making the above three components explicit was a highly iterative process which involved finding the right grain size for identifying learning problems as well as identifying machine learning requirements and capabilities.

Table 4. Examples From Domain and System Templates

Domain Template	System Template
Example1	
The knowledge for the learning problem can be characterized as: a) Based on examples b) Based on examples and their solutions c) A poorly structured collection of many isolated facts d) A complex, highly structured collection of facts e) Models of structure and function f) Concise, unified theory	How does the learning system represent a learning problem? a) Features vectors Yes No b) First order logic Yes No c) Distributed network Yes No d) Production system Yes No e) Semantic network Yes No f) Frames/scripts Yes No g) Frames with prodecural attachment Yes No
Example 2	
How large is the (classification) problem domain? Number of classes of objects: Number of observations per class: Number of properties per object: Number of values per property:	Quantitative capabilities of learning system Number of concepts: Range of observations per concept: Range of properties per observation: Range of values per property:
Example 3	
Input in the learning problem is a) Numeric b) Symbolic Interpretation of input depends on: a) Only its current value b) Its trend or time course (e.g., it may be important to know whether a measurement is rising or falling) c) Values of other inputs (e.g., interpretation of a patient's contextual symptoms may require knowledge about whether the patient is in critical or stable contition) d) Having followed the input over time such that a customized set of expectations is developed (e.g., a specific user style)	The semantics of the representation concern: a) Quantitative information b) Qualitative information The system can interpret input that depends on: a) Only its current value b) Its trend or time course c) Values of other inputs d) Having followed the input over time such that a customized set of expectations is developed
Example 4	
There is a strong bias in the learning problem to avoid: False alarms Misses Neither Both NA	The system makes use of: Positive examples Negative examples Both

VS9.10.2

We do not expect that this iterative process will ever be completely avoidable. We do, however, hope that by explicitly identifying and reporting the process we went through and the issues we uncovered, that future selections of appropriate learning problems and approaches will become more straightforward.

The first problem we describe is for an expert system for the diagnosis of aircraft malfunctions. The performance problem is to diagnose these malfunctions from a given set of aircraft symptoms. We will assume that this will be achieved by reasoning directly with the symptom-malfunction pairs (rather than reasoning causally with a device model).The learning problem is to acquire a general representation of system malfunctions, and a decision process for diagnosing the underlying malfunctions that account for the set of symptoms. We will also assume the availability of numerous examples of symptoms and their associated malfunction classification.

To obtain the results discussed below, we began with a candidate set of systems that provided the knowledge required for solving a particular learning problem. We eliminated a candidate learning system when it fails to address one of the critical learning problem properties identified in the domain template (e.g., fails to learn from negative examples). We also eliminated a candidate learning system when it required domain properties that were not available in the learning problem as we defined it (e.g., required a domain theory that was not available).

Figure 2 contains the decisions we made in identifying a machine learning system for the diagnosis problem. We first considered the knowledge requirement that must be learned--classification knowledge and a reasoing procedure. On this basis we eliminated the machine learning problem solving and planning systems, which do not accommodate this classification problem very well. Next we considered the amount of knowledge available to support an explicit domain theory for reasoning about the causal relationship between symptoms and faults, because explanation based systems depend on such a theory. We eliminated the explanation based systems and ACM (which incorporates a domain theory implicitly) due to the lack of such knowledge for this problem. We next eliminated the Version Space algorithm and ID3 because we anticipated the need for multichotomous categories. Finally, we considered the number of experiences that would be available for this problem. We selected AQ11 as the system that would probably require fewer experiences and less time.

The second problem we will consider is for an expert system that displays information to a user--while balancing a number of 'soft-constraints'. The performance problem is to present information in a way that is most advantageous to a viewer. As an example of such constraints, certain information may be more critical than other information: human factors principles may suggest that it be placed in a certain location to ensure that the user notices it. However, another human factors principle might suggest that the same information should appear in the same location all of the time. Furthermore, some information may require particular display capabilities—it may be pictorial, graphical, or numeric. In short, there are a number of potentially conflicting constraints that govern the arrangement of information for display. The learning problem is to modify a correct but inefficient search procedure to arrange information on the display device(s).

We assume the availability of an inefficient search procedure that respects soft constraints. In this procedure constraint addressed as a subgoal of the information arrangement problem. We also assume the availability of a database of information arrangement tasks and a problem solving system. The learning experience inputs consist of a problem solving tree that terminates in successful information arrangement, i.e., correctly assigns

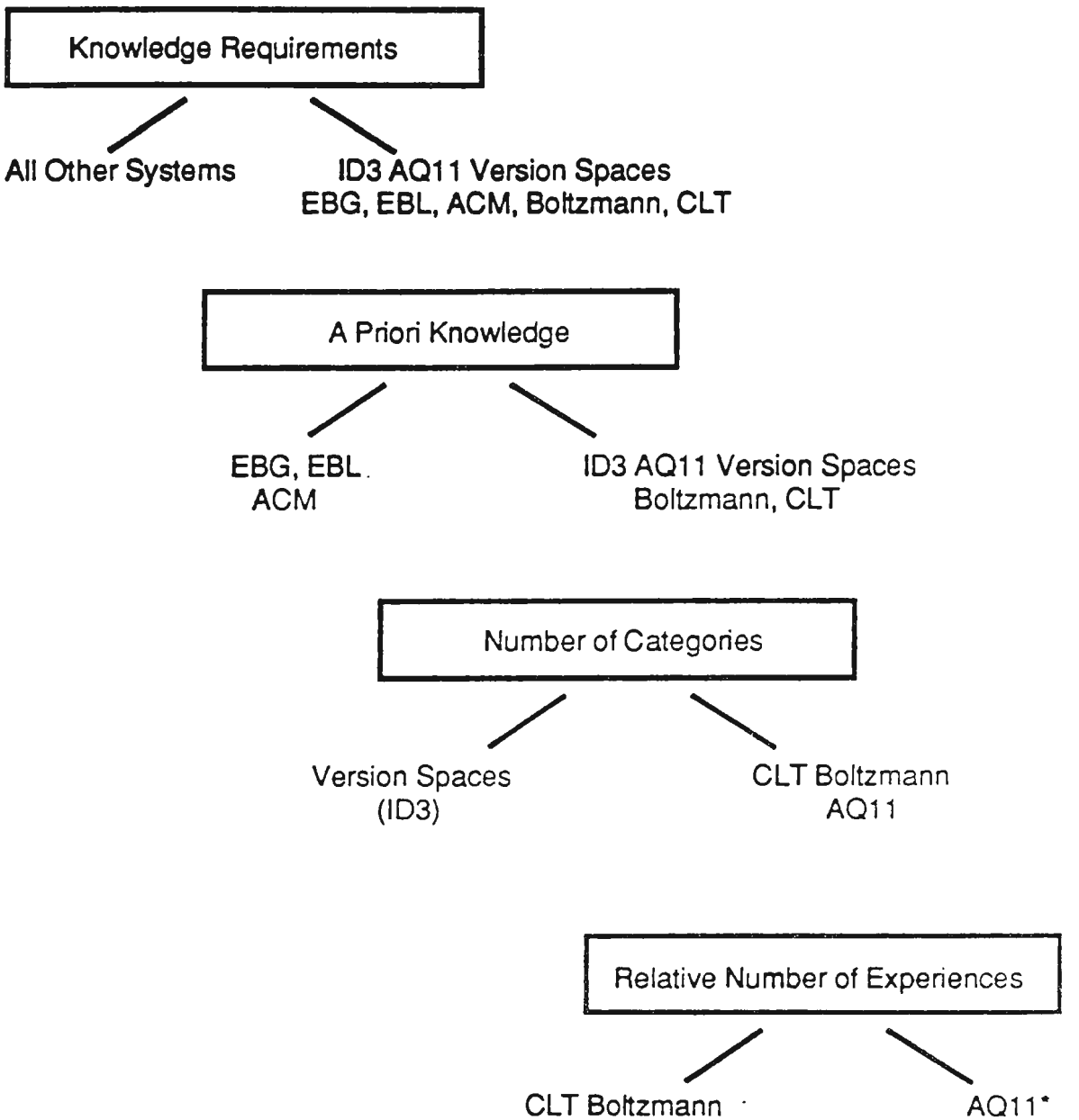


Figure 2. Diagnose aircraft malfunction

information for display. The goal of the learning is to produce a search procedure that ignores bad branches in search space by knowing when to apply its operators.

Figure 3 contains the decisions that we made in identifying a machine learning approach for the Information Display Search problem. Again, we first considered the knowledge requirement of the learning problem--optimizing the search for a correct display by learning applicability conditions--and this time selected the problem solving systems. Next, we considered the number of experiences that would be available, and that would be required for each candidate machine learning system. We eliminated the similarity based systems because we believed that the combinatorics of information for display would be too large for these approaches. We eliminated STRIPS, LEX, and ACM based on the judgment that the problem has a subgoal structure (inherent in the multiple design constraints). Systems like STRIPS, LEX and ACM would probably have difficulty properly representing the applicability conditions of rules for this domain (i.e., the subgoal context). This risks an unsuccessful search for correct applicability conditions to optimize the display arrangement procedure. Checkers also lacks an explicit goal--subgoal structure. Its evaluation function for a given operator does not take a changing subgoal context into account. (A Checkers application here would also be a stretch for other reasons). Thus, having eliminated Checkers, the remaining candidate systems were EBL and SOAR. We chose SOAR because this learning problem involves executing the problem solving task. SOAR clearly supports problem solving. We are uncertain of EBL's current capabilities as a functioning problem solver.

CURRENT AND FUTURE WORK

The analysis presented above is a modest beginning in attempting to understand the issues involved in using machine learning for knowledge acquisition. Much work remains, directed towards both the dependencies themselves, and issues not yet addressed by the dependencies. For example, we are interested in using the dependencies in a more systematic fashion. In particular, we have implemented a simple algorithm that calculates the match or fit between a system and a learning problem by examining the dependencies that are violated and supported by a learning problem--machine learning pairing. However, there are a number of problems with applying the dependencies to find good matches between machine learning systems and learning problems. One problem is that we have only examined dependencies between the learning domain and the other machine learning components. There are certainly dependencies between any pair of components. Dependencies may interact. This fact may prohibit their systematic use to arbitrarily select or construct a machine learning system for a given learning problem. Thus, the quantitative aspects of our dependency analysis remain work in progress.

Another large gap is in the area of abstracting a learning problem and integrating the results of the abstracted problem with the existing knowledge base of an expert system. Mitchell (1987) noted some of the problems that he faced when building learning systems that assist performance systems. For example, he designed the learning apprentice system LEAP (Mitchell, Mahadevan, and Steinberg, 1985) which learns rules that are used by an expert system for circuit design. The rules that LEAP learned were incompatible with those used by the expert system. Specifically, the rules used by the expert system contained control knowledge as well as factual knowledge for designing circuits whereas LEAP learned rules consisting only of factual knowledge. A rule translator was designed to solve this incompatibility problem.

In the near future, we expect to be integrating current machine learning technology with existing pilot-aiding expert systems. These systems aid pilots by reducing workload and

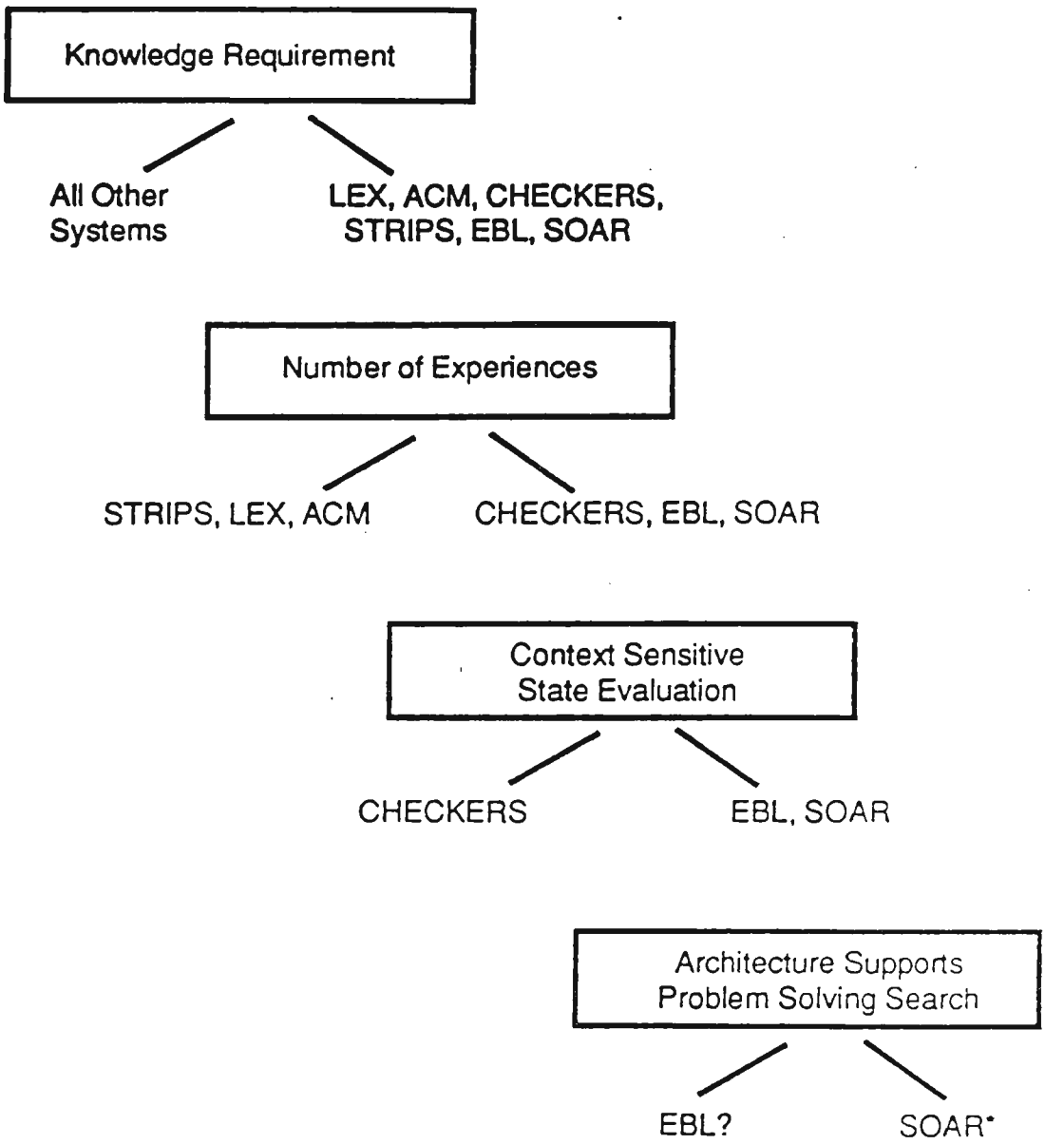


Figure 3 . Optimize Display Search

increasing the pilot's situation awareness. Here, we will develop a learning module that is invoked when a pilot-aiding system fails in its performance. Typically, these failures result from an incomplete or inconsistent knowledge base. We face a number of obvious integration issues. For example, the learning module must acquire knowledge in a form that is useable by the pilot-aiding system. At the implementation level, we must design a module that produces the same kind of data structures as those used by the pilot-aiding systems. In addition, the module must be able to integrate these data structures into the existing knowledge base of the pilot-aiding system without inappropriately disrupting the existing functionality of the system.

These examples illustrate attempts at applying machine learning technology to existing performance systems. We believe, however, that performance system should be designed at the outset to include a learning component. Ideally, a knowledge-based expert system will be closely integrated or coupled with the learning component. There are several clear benefits to constructing a system in such a manner. First, the experiences that are input to the learning system are more likely to be meaningful. Learning experiences obtained during performance typically focus the machine learning system on deriving representations that increase the efficiency of the performance component (as in LEX, see also SOAR and ACM). Thus, the experiences focus the system on learning something that is clearly useful. These experiences contrast with those of systems whose learning mechanisms are not embedded in performance systems. It is not always the case that the experiences of these systems, and therefore the learning, will be useful. For example, while some of the experiences of EURISKO allow the system to derive useful representations, others may actually disrupt the system's functioning (see Lenat, 1983).

Second, performance systems enable the development of clear criteria for evaluating the representation derived by the learning component. The evaluation of a learning system that is embedded in a performance system depends on how successful the performance system is in solving a given task. Here, representations are evaluated with regard to whether they contribute to the improvement of the performance system. While other, more theoretically oriented criteria are also important, for example, understandability, coherence, etc., they are easier to appreciate when they can be tied to an improvement of the performance system.

By integrating machine learning systems with knowledge-based systems, we come closer to achieving a more principled design and construction of knowledge bases. First, knowledge acquisition is based on a definable learning algorithm rather than the ill-defined interaction of a domain expert and knowledge engineer. The knowledge engineer will know why and how knowledge was added to the system. Second, the design of continually adaptive automated systems will force us to consider the initial knowledge base design and content more carefully, making sure that the knowledge is explicit and accessible for modification.

SUMMARY

To summarize, we have presented a general framework for comparatively evaluating machine learning programs. Using the framework as a basis, we have developed questions that characterize machine learning programs and their possible application domains. We argued that dependencies exist between features of a domain and features of a machine learning system's components. Using these dependencies, one can determine the potential relevance of a given machine learning system as a knowledge acquisition aid in a domain. To justify this argument, we contrasted the relative applicability of some machine learning systems for various domains. We see our approach as an important step towards assessing machine learning technology for knowledge acquisition in knowledge-based systems.

REFERENCES

Davis, R., Interactive transfer of expertise: Acquisition of new inference rules, *Artificial Intelligence*, 12, 121-157, 1979.

Dietterich, T. B. Learning at the knowlede level. *Machine Learning* Vol. 1 (3) 287-315, 1986.

Greeno, J. G. & Simon, H. A., Problem solving and reasoning, In (Eds.), R. C. Atkinson, R. J. Herrnstein, G. Lindsey & R. D. Luce, *Steven's Handbook of Experimental Psychology* . New York, NY: Wiley, 1986.

Hayes-Roth, F, Waterman, D.A., and Lenat, D.B. (Eds.), *Building Expert Systems*, Reading, Mass., Addison-Wesley, 1983.

Kline, P.J. and Dolins, S.B. Choosing architectures for expert systems, {\em CCSC Technical Report 85-01-001}, Rome Air Development Center, Griffis AFB, New York, 1985.

Laird, J.E., Rosenbloom, P.S., and Newell, A., Chunking in Soar: The anatomy of a general learning mechanism, *Machine Learning*, 1, 11-46, 1986.

Lenat, D.B. EURISKO: A program that learns new heuristics and domain concepts. *Artificial Intelligence* , 21. 61-98, 1983.

Michalski, R.S., statement made at First Annual Conference on Knowledge Acquisition for Knowledge-based Systems.

Michalski, R.S. Understanding the nature of 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.

Mitchell, T. M. Remark from Symposium on Applications of Machine Learning to Pilot Aiding. Unviersity of California, Irvine, June, 1987.

Mitchell, T.M., Mahadevan, S., and Steinberg, L.I., LEAP: A learning apprentice system for VLSI design, *LCSR Technical Report 64,* Rutgers University, Dept. of Computer Science, New Brunswick, NJ.

Mitchell, T.M., Utgoff, P.E., and Banerji, R., Learning by experimentation: Acquiring and modifying problem-solving heuristics, {\em LCSR-TR-31}, Rutgers University, 1982.

Newell, A. The knowledge level, *Artificial Intelligence.* 18, 87-127, 1982.

Scott, P.D., On the systematics of machine learning, *In Preparation*..

Scott, P.D., Knowledge-oriented learning. *Proceedings of the 8th IJCAI*, Morgan Kaufmann, Los Altos, CA 432-435, 1983.

Stefik, M., Aikens, J., Balzar, R., Benoit, J., Birnbaum, L., Hayes-Roth, F., and Sacerdoti, E., Basic concepts for building expert systems, In Hayes-Roth, F., Waterman, D.A., and Lenat, D.B. (Eds.), *Building Expert Systems*, 1983.

Smith, R.G., Winston, H.A., Mitchell, T.M., and Buchanan, B.G., Representation and use of explicit justification for knowledge base refinement, In *Proceedings of the Ninth Annual International Joint Conference on Artificial Intelligence*, pages 673-680, August, 1985.

Winston, P.H., Learning structural descriptions from examples, In Winston, P.H., (Ed.), *The Psychology of Computer Vision*, 157-210, 1975.

Wisniewski, E.J., Winston, H.A., Smith, R.G., and Kleyn, M. A conceptual clustering program for rule generation, To Appear In, *International Journal of Man-Machine Studies*.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

Validation of a Knowledge Support System

Mildred L. G. Shaw & J. Brian Woodward
Department of Computer Science,
University of Calgary
Calgary, Alberta,
Canada T2N 1N4

ABSTRACT

At the previous workshop on Knowledge Acquisition for Knowledge-Based Systems in 1986, criteria for a Knowledge Support System (KSS) were discussed, and a preliminary version of KITTEN (Knowledge Initiation and Transfer Tools for Experts and Novices) was described and demonstrated on Apollo workstations. A three-stage model for KSS evaluation and validation is discussed. KITTEN has been evaluated against the first stage of the model and the results are reported in the two domains of spatial interpolation techniques to produce contour maps and in trouble-shooting and maintenance of valves for oil and gas pipelines. Some preliminary results are described on validation experiments to show the extent to which this system can replace standard interviewing techniques. Some of the question asked were:

1) To what extent does an expert find the generated rules meaningful?
2) To what extent do other experts agree with the expert's ratings of her/his elements on her/his constructs?
3) Can the same constructs be obtained from the expert after a significant time lapse?
4) Can the same ratings of the elements on the constructs be obtained from the expert after a significant time lapse?

INTRODUCTION

In the last Workshop on Knowledge Acquisition for Knowledge-Based Systems, in it was clear that several researchers were considering the great potential of the repertory grid (Kelly, 1955) as a tool for knowledge acquisition, and as a source of rules for a rule-based expert system. Shaw and Gaines (1983, 1986a) have put great emphasis on the need to be able to validate any technique for knowledge engineering. It is not sufficient to show that reasonable expert systems can be developed. One must attempt to evaluate the accuracy and completeness of the knowledge transfer. This is not an easy task because there are few well-established domains of expertise where the accuracy of the elicitation can be tested. A study has been carried out using PLANET(Shaw 1982) to determine whether PEGASUS can be used to elicit the Business Information Analysis Integration Technique (Carlson 1979, Sowa 1984) distinctions from those with some accounting or business knowledge (Shaw & Gaines 1983). The experiments have been based on taking a spectrum of those with assumed expertise in business record keeping, from businessmen and accountants, to accounting students at various stages of their education, and using PEGASUS to elicit their constructs relating to business transactions. Again, in the interests of an initial open-ended approach, the construing was not tightly constrained to relate to the record-keeping aspects of the transactions.

Various forms of analysis were applied to the resulting grids. The constructs elicited were compared with the BIAIT constructs by analysing with ENTAIL the compound grids obtained by merging the constructs elicited from each person with those corresponding to the BIAIT

constructs. This showed that all the BIAIT constructs exist within those elicited. The set of grids obtained was also analysed with SOCIOGRIDS to see whether the variations of expertise across the group of subjects used shows up clearly. This is a good test of the basic validity of the technique and its foundations in the notion of construing. For example, in an earlier study (Shaw 1980) of quality control for a garment company the SOCIOGRIDS analysis re-constructed the management hierarchy of the firm from the construct systems of staff involved in construing garment faults.

EVALUATION OF KNOWLEDGE-BASED TOOLS

Evaluation studies of knowledge-based systems (specifically expert systems) have been reported and methods for validation have been outlined. A typical example is Siegel (1986) who points out the need to evaluate the output of expert systems and he outlines in good detail the methods for accomplishing this task. He includes criterion variables and the methods for testing. These methods are based on the use of test cases as a developmental tool through successive improvements of the system. Test cases provide standard input to the system and output can be tested for consistency.

Until the relatively recent development of knowledge acquisition tools, the evaluation of knowledge-based systems focused of the performance of the system against the problem it was designed to solve. Knowledge acquisition was considered difficult to design and problematic at best. Waterman (1986) views the iterative development of expert systems as a method to compensate for less than effective knowledge acquisition procedures. The expert system, itself, acts as a check on the acquisition procedures. However, criticisms of this approach have been voiced (Gaines, 1987). Subjective validation of the expert system's knowledge-base and performance by the expert may provide inaccurate or biased criteria (Cleaves, 1986) or may point out only the components which do not "work" but may say little about what does work. Gaines suggests that objective performance standards be developed and for knowledge-based systems and acquisition tools be evaluated against these standards. Knowledge relevancy, accuracy, consistency and completeness have been posited as objective performance criteria.

Reports of evaluation activity for automated knowledge acquisition tools usually appear only as tangential comments in papers outlining the design and development of a knowledge-based system. A notable exception is Michalski (1983) who compared the inductively derived rules from AQ11 with direct representations of the expert's knowledge. Other automated systems such as MORE (Kahn et al, 1985), SALT (Marcus, 1986) and MOLE (Eshelman et al, 1987), which have been designed as as automated interviewers for specialized knowledge-bases, approach knowledge-base criteria through a "successive refinement" method. Although this does not constitute a strict validation of the acquisition tool itself, the refinement process helps to build the integrity of the knowledge-base. This can be considered as one method of validating the knowledge acquisition tool.

Burton et al (1987) have recently reported an evaluation study of a number of knowledge acquisition techniques: interview, protocol analysis, goal decomposition, and card sort in the domain of rock identification. They took 32 undergraduates as subjects and looked at time taken for elicitation, time taken to code the transcript into production rules, number of rules elicited (correct or incorrect), number of clauses constituting the elicited rules, and completeness of the rule set. They also measured the personality variables introvert or extravert, and field dependent or independent.

Some of the closest work to actual knowledge validation as well as validation of acquisition tools appears in the investigation of expertise. Johnson (1986a, 1986b) has established a research procedure which attempts to validate the expertise required to solve problems. Using structured

interviews, verbal thinking-aloud protocols and protocol analysis based on systematic observations of problem-solving behaviour, Johnson models the operative knowledge of a task. His empirical testing of events includes the use of a panel of experts to validate results of knowledge acquisition as well as the use of test cases which are both similar and dissimilar to the original problem-solving task.

Although many knowledge acquisition systems have been designed, little has been reported on the evaluation of the systems per se. What appears lacking is a useful framework to guide acquisition tool evaluation. The following section outlines a model for the evaluation and validation of knowledge acquisition systems.

FRAMEWORK FOR EVALUATION

In discipline research the dependent variable acts as the starting point and various factors are hypothesized which might predict that variable. In evaluation research, the independent variable (e.g. the program, the system, the procedure) acts as the starting point and assessments are made against a set of criteria or goals (Bernstein, 1976). Evaluation research is expected to tell how well something "works": how well something does what it is supposed or conceived to do. Evaluation can also be structured to tell when, why and how something works.

In order to be effective, evaluation designs require clear explication of components and requirements. The first is to determine the external criteria which act as comparison points for the system under evaluation. A priori specification of these criteria ensures rigor and clear measures of the system's performance. Establishing a useful evaluation framework also requires the recognition of various levels of operation and measurement in the external criteria. This represents a second requirement.

Specific determination and description of the input and output of the system under evaluation is a third requirement. The methods and procedures used to measure the inputs and outputs of the system require simplicity and accuracy. As well, the methods of processing inputs to outputs (e.g. the algorithms) require clarification, justification and theoretical support. This fourth requirement, together with the third, allows the developer or researcher to use evaluation results not only to establish the system's usefulness of operative effectiveness but also to help guide future development and improvement of the system under evaluation.

A fifth requirement of evaluation research is the proper selection of populations for study. In many cases the target population has been considered in the design of the system. Samples drawn from this population for research purposes cannot pose a threat to internal and external validity (Anderson, 1975).

The final requirement reflects the number of measurements taken of the system during evaluation activity. A multiple measurement schedule built into the evaluation design ensures that reliability can be strengthened and provides necessary information for development purposes and for assessing the degree to which the system meets performance criteria.

This section outlines an evaluation model for Knowledge Support Systems (KSSs). Applying the above criteria to KSSs, evaluation stages and specific evaluation tasks are outlined for the evaluation activity. General design considerations have been elucidated for KSSs (Shaw and Gaines, 1986b). These are also incorporated into the evaluation framework.

The evaluation model in Figure 1 presents three distinct levels of activity. These distinctions represent the various levels of operation of KSSs. Primary Level evaluation is concerned with

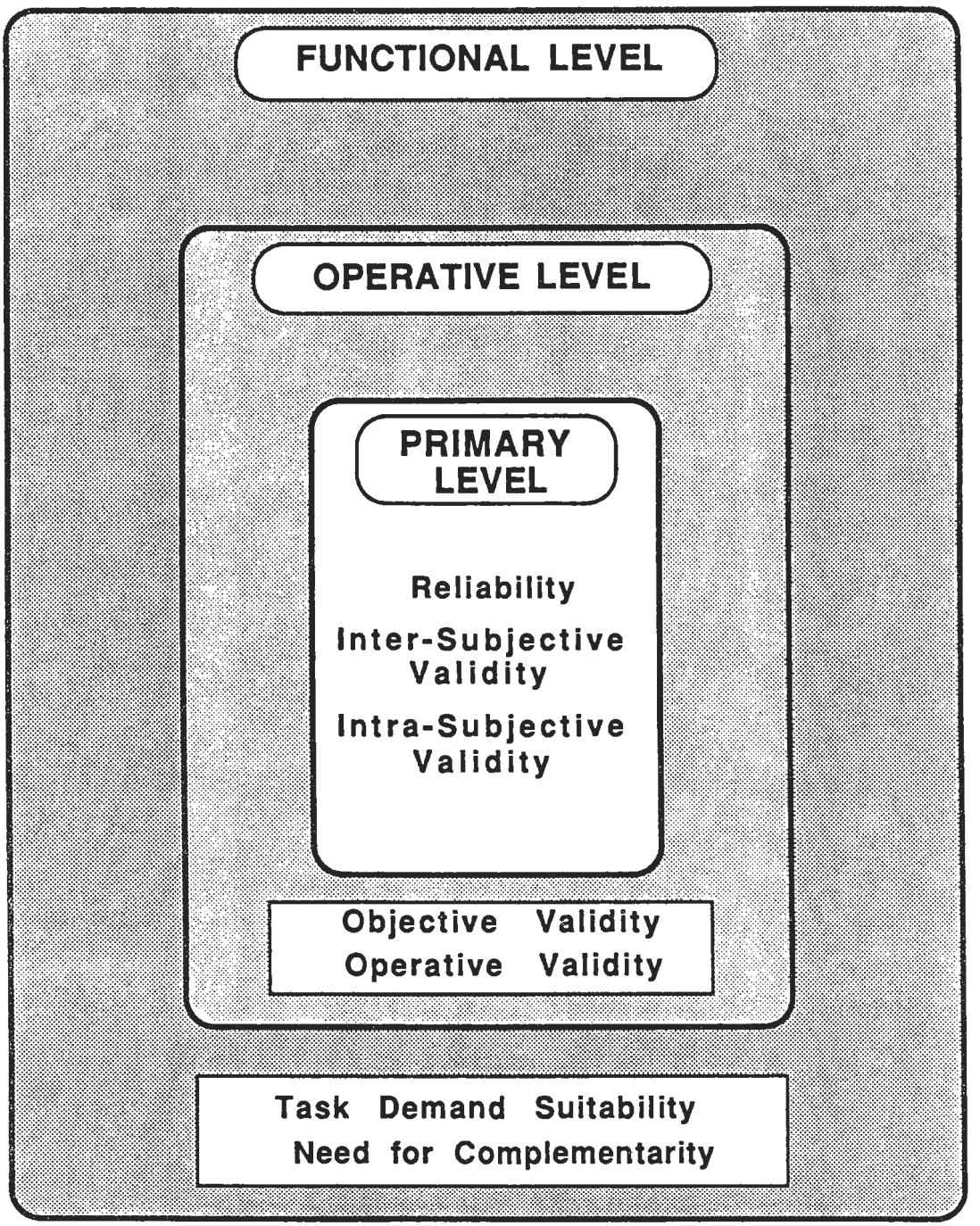


Figure 1. Evaluation Model for Knowledge Support Systems

very basic attributes of KSS operation: the systems reliability over time and the degree to which system output demonstrates basic measures of validity.

Primary Level

A measure of diachronic reliability (Chow, 1987) appears most suitable for evaluating KSSs. This measure reflects the degree of similarity, over time, of the input required for the system and of the calculated output. A KSS reflects a high degree of reliability in the sense that given the same, specific task demands, similar information would be required as input to the system on separate occasions. Also, the output of the KSS, the results of the algorithms, would reflect a high degree of similarity over time. Strictly speaking, this approach to reliability is strongly influenced by the reliability of the user of the KSS. However, the KSS can be viewed as an aid to the user which helps the user maintain consistency over time given the same task demands. Also, KSSs are not used in isolation but are used by individuals so any measure of a KSS must be done with those would would use the system. If reliability measures are low, the system cannot be considered consistent or helpful. For a KSS, this could indicate a wide variance in input and output which would affect any subsequent procedures or uses.

Although **reliability** stands as a necessary requirement of any KSS, other basic requirements demand attention. The KSS must also demonstrate various forms of **validity**. In this sense, validity refers to the extent to which the KSS produces what it is supposed to produce. At the Primary level of evaluation, validity is defined by two measures. The first is **intra-subjective validity**. Based on the concepts of content and face validity (Travers, 1969; Campbell & Stanley, 1963), this type of validity is defined as the extent to which the KSS output is a representative sample of that output. Valid KSSs produce output which has a high degree of similarity with what is apparently expected. The absence of this form of validity indicates that the system may be performing its calculation incorrectly and/or that the output bears little resemblance to what would be expected by the designer, by the user of the tool or by the domain expert. This result has serious implications for any subsequent use of KSS output.

Another, complementary measure of validity at this level of evaluation is that of **inter-subjective validity**. Based on the concept of concurrent validity (Chow, 1987; Travers, 1969) this type of validity is a measure of prediction of events which occur at roughly the same time. If two or more individuals, with similar experience in an area, produce similar output using a KSS, then the system can be said to afford a degree of inter-subjective validity. The extent to which two or more people produce the the same output represents the degree of validity.

Even though experts may seldom agree, if a task is properly defined, specifically outlined and agreed upon as important, then a certain degree of similarity can be expected in the outcome of the KSS. This suggests the selection and development of a set of test cases with specific task requirements. These test cases can then be used to determine the degree of inter-subjective validity demonstrated by various experts, while using the system. In the same way as for reliability, the KSS can be seen as an aid to users.

Operative Level

Once the KSS has demonstrated acceptable levels of reliability and intra- and inter- subjective validity, it can be evaluated against more objective criteria. **Objective validation** refers to the degree to which the output of a KSS agrees with well established facts in the domain. This can be accomplished by measuring the degree of convergence of KSS output with established facts from reputable experts and or from reputable printed material. Also, a measure of the degree of divergence of KSS output from known non-truths can be ascertained: the degree to which the KSS output is dissimilar from known falacies. Valid KSS output demonstrates a high degree of

similarity (convergence) with established domain facts and a high degree of dissimilarity (divergence) from falacies.

Another form of validation at this level concerns the usefulness of the output of the KSS. **Operative validity** is defined as that knowledge which is necessary for accomplishing a task or class of tasks (Johnson, 1986). By definition, operative knowledge is that body of information which is necessary and sufficient to meet task demands. It suggests that any knowledge which is unnecessary for the task is considered tangential and/or superfluous and should be eliminated from consideration. Only the knowledge which is used to solve the problem need be considered. The degree to which a KSS is able to reduce the amount of unnecessary or superfluous output represents a measure of operative validity.

Functional Level

Once the KSS has demonstrated acceptable degrees of objective and operative validity, its functional role attributes require assessment. In other words, the KSS does what it is supposed to do but in order to be used effectively or for its use to be generalized, constraints and caveats on its use need elucidation and study. The usability (Gaines, 1987) of the system requires assessment. This refers to the ease of use and the understandability of the knowledge acquisition tool by the expert. The KSS requires testing using a variety of task demands apart from those used in the last level of evaluation. Measures of the degree of KSS reliability and validity on similar task demands and on dissimilar task demands begins to indicate the limits of the system's usefulness, effectiveness and areas of operation.

At this level of evaluation it is necessary to elucidate any and all support processes or complementary procedures which are required prior to or after using the KSS. Effective use of the KSS may require support to meet its general requirements. Shaw & Gaines (1986b) have outlined these general requirements for a KSS.

METHODOLOGY

Figure 2 outlines the procedural steps and the data collection points used in the Primary Level evaluation of KSS0 which is the grid elicitation and analysis part of KITTEN (Shaw & Gaines, 1986b). The version used runs on the Apple Macintosh. The Figure shows the first step as Procedure 1 (P1) which introduced the subjects to the requirements of KSS0. At this time a specific task was agreed upon and a purpose for the grid elicitation was developed. The first data collection point (G1) consisted of the elicitation of a grid by each subject individually. This provided an initial set of elements and constructs particular to each subject. The second collection point (G2) consisted of the subjects exchanging their first grids (elements and constructs) to produce other grids. The next step was a procedure (P2) to have the subjects agree on a common set of elements. This common element set provided the basis for the next data collection point (G3) which consisted of the elicitation of a grid by each expert using common elements. This elicitation was done, as the other, separately for each expert. P3 represents a wait of over two months before the next data collection point (G4). At this time a repeat of the activity at G3 was completed using a common element set but the regeneration of constructs. For G5 the subjects were given the element and construct set from G3 and asked to re-rate the elements. The final data collection point (P4) consisted of giving the subjects a selected list of entailments using the ENTAIL algorithm (Gaines & Shaw 1986) generated from the G3 grid. A set of examples from four separate levels of significance of entailment were selected and randomly presented to the subjects who were asked to rate each entailment on a four point scale from correct to incorrect.

Event	Description
P1	determine task demands and elicitation purpose
G1	grid elicitation using own element set
G2	grid eliciation using other expert's elements and constructs
P2	experts discuss and produce a common element set
G3	grid elicitation using common element set
P3	waiting period: 2 to 3 months
G4	grid elicitation using G3 element set
G5	grid elicitation using elements and constructs from G3
P4	obtain subjective ratings from entailment output from G3
	Analysis

Figure 2. Evaluation Design Table

The Studies

Two separate studies were completed using specific task demands from two separate and quite different domains. The first study involved three researchers in a university geography department each of whom is an expert working in the field of spatial interpolation techniques to produce contour maps. Each was interviewed separately (P1) and was involved independently for an hour or more in the first session on the task of "evaluating spatial mapping techniques" (G1). In the second session, on the following day, each expert exchanged his or her constructs with the others. This involved Expert 1 in seeing Expert 2's elements and constructs without the ratings, and rating them as he would have done (G2a). He repeated this for Expert 3's grid (G2b). This was done for each expert in turn.

Following this session, all three experts met together to discuss the elements each had used, and to agree both on the meanings of each, and on a common set that all could use. There were twelve of these decided after 30-40 minutes. In the third session, in the same week, each expert elicited another set of constructs which could include the previous ones elicited, as well as any he or she had seen when doing the exchange procedure. This was G3. Three constructs were added at this stage by the experimenter: **very important -- not very important; very effective -- not very effective;** and **widely used -- not widely used.** Each expert also rated the constructs for importance on a 1 to 10 scale: 1 most important, to 10 least important. After this session, the experts were asked not to think about the topic more than they usually would, not to discuss the topic with each other, and not to review their constructs before the next session.

In the fourth session, twelve weeks later, each expert in turn elicited another set of constructs using the agreed set of elements from session three producing G4. Immediately this was completed, in about an hour, each expert was shown his elements and constructs from session three, that is, G3 with the ratings removed, and asked to re-rate these on the same scale. This gave G5.

The entailments were produced for each expert from G3. These rules were then sampled from each category of significance, $\geq .15$, .14 to .10, and .09 to .05 and randomly presented to the expert (P4) who was asked to rate them on a scale 1- correct, 2 - likely to be correct, 3 - not likely to be correct, and 4 - incorrect.

In the first session, two of the experts used a 5-point scale, and one a 9-point. When agreeing on their common set of elements, they also agreed to use a 9-point scale. Although this seems cumbersome when using numbers it is, in fact, a finer discrimination of a 5-point scale, having all the in-between points available for finer distinctions. All three experts agreed that the system was easy to use, allowed easy correction of errors, and gave them the options that they needed to peruse the data.

In the second study, a trainer and an engineer in the field of gas pipeline valve maintenance provided their expertise. These two individuals were chosen because they were both considered knowledgeable in the area, but also because they brought a different background to the field: one from a field-operator perspective, the other from that of an engineer. The trainer had been involved in training gas pipeline operators in proper maintenance of pipeline valves for 15 years and the engineer owned a company which sold valves. Each was interviewed separately (P1) and a diagram of the area was developed. Common to each diagram was a crucial step of determining the type and make of the valve. This step was necessary because knowledge of the valve type and manufacturer gave important clues to the operator concerning proper maintenance. The task demand was defined as " the determination of valve characteristics". Each subject completed the design requirements except for P4.

The Measures

Operational definitions for the various reliability and validity measures were used to evaluate the KSS. **Construct reliability, R_c,** was defined as the degree of similarity in constructs generated in G3 and G4. Operationally, this measure clustered the sets of constructs from both grids using the FOCUS algorithm (Shaw, 1980) and measuring how many clusters above a cut-off of 75% similarity were mixed. Those constructs which were matched on the G3 and G4 grids at 75 % or better were compared to the total number of possible matches between G3 and G4. Figure 3 shows how the calculation was made for expert E2 from study 1. The following formula was used to calculate R_c for each subject:

$$R_c = \frac{\text{(number of clusters from both G3 and G4)}}{\text{(total number of clusters)}} \times 100$$

Rating reliability, R_r, was defined as the degree of similarity in element ratings in G3 and G5. To calculate this measure, the MINUS algorithm (Shaw, 1980) was used to subtract two grids which have the same elements and constructs. The result was a percentage degree of difference between the two grids. The following formula was used to calculate R_r for each subject:

$$R_r = (100 - \text{difference percentage in G3 and G5})$$

Inter-subjective validity for constructs, V_c, was defined as the similarity in constructs in G3 for each of the experts. The grids for each expert at G3 were combined, and the same measure used as described in construct reliability. In order to be matched, the clusters from study 1 were counted as mixed if they contained constructs from two or three experts.

$$V_c = \frac{\text{(number of clusters from all G3s)}}{\text{(total number of clusters)}} \times 100$$

Inter-subjective validity for ratings, V_r, was investigated by looking at the **exchange grids**. Do the experts see the topic in a similar way at the same time? By exchanging elements and constructs they are able to view the topic from the perspective of the other. To do this involves an understanding of what the other's elements and constructs actually **mean** in the domain, or **can I construct a point of view which makes sense of them?** Not only that, but **do I agree with the other's construction of the topic, having understood the perspective of the other?** This was defined as the degree of similarity between one subject's ratings of his/her own elements and constructs and another subjects's ratings of the first subjects's elements and constructs. Operationally, this measure was calculated using the MINUS algorithm to subtract one grid from another resulting in a difference score. The following formula was used:

$$V_r = (100 - \text{difference percentage in G1 and G2})$$

[N.B. In study 1 there were two G2s: G2a and G2b, so two measures resulted.]

Finally, **intra-subjective validity, V_i,** was defined as the degree of similarity of the rules produced by the ENTAIL algorithm with what the expert apparently expected. The ratings of these statements at P4 were then compared to the entailment output from G3. The number of identical ratings and the number of similar ratings (differing by only one value) were selected. The following formula was used to determine V_i:

$$V_i = \frac{\text{(number of same ratings + number of similar ratings)}}{\text{(total number of rules given)}} \times 100$$

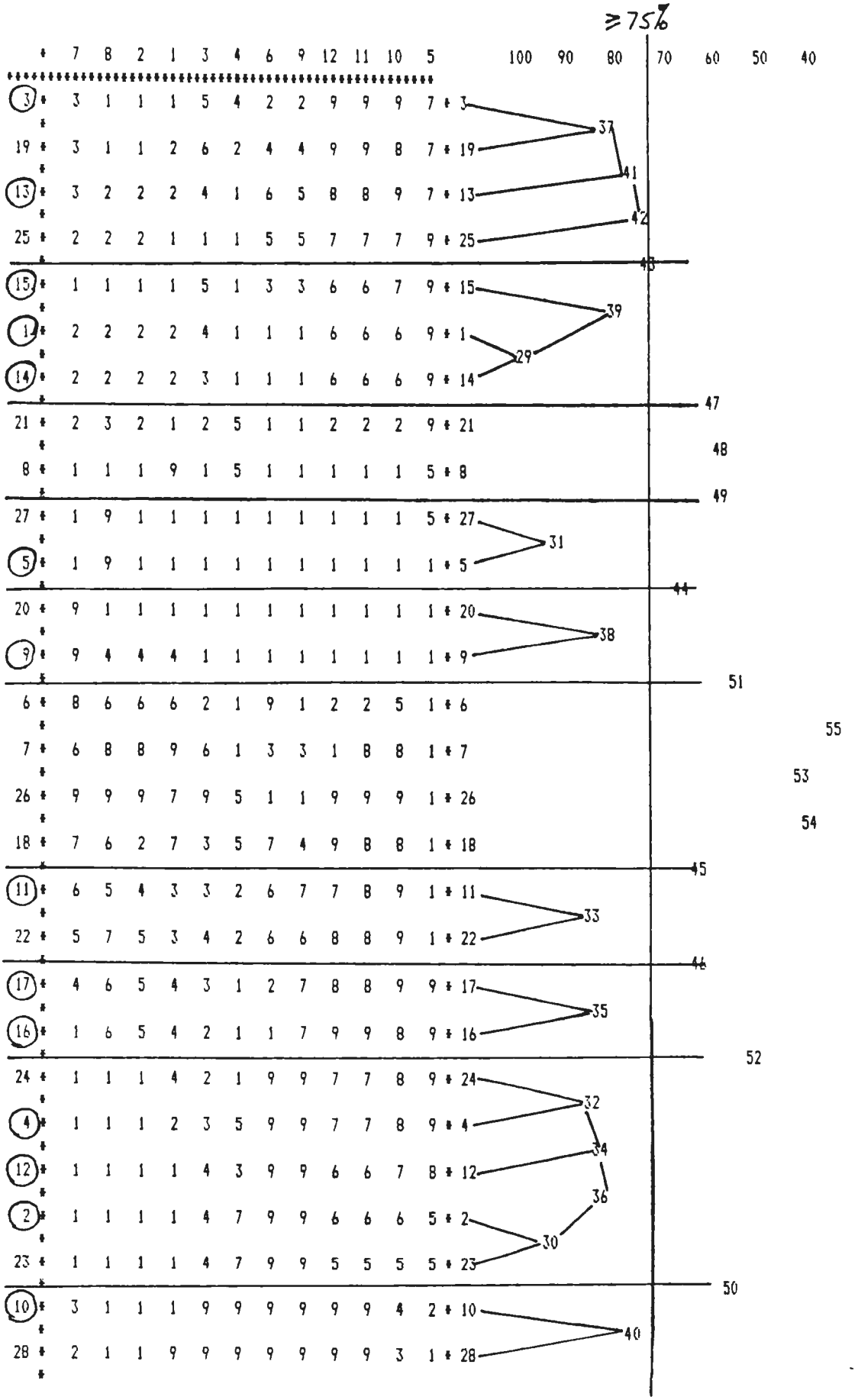


Figure 3. Construct reliability = (number of clusters from both G3 and G4) x 100
(total number of clusters)

= 6 / 8 x 100 = 75%

RESULTS

Figures 4 to 6 display the results from the two studies. The **construct reliability** scores displayed in Figure 4 ranged from 50% to 83% in the first study, but were 100% in the second. This may reflect the fact that the three experts in the first study are academic researchers at a university, and hence tended to be very provisional, exploratory and accepting of different ways of considering the problem. They were at no time fixed in their views of the topic. In the second study, however, the experts were professional using their expertise in the field, and hence much more decisive and sure of what they were doing and how it should be done. The **rating reliability** scores also displayed in Figure 4 ranged in both studies from 84% to 89%, indicating that all experts were using their ideas in much the same way from one occasion to the other and that KSS0 was able to reflect this consistency.

Inter-subjective validation scores, displayed in Figure 5, that is to what extent the experts agree, range in the first study from 68% to 76%. It is interesting to note that the lowest scores of 68% both come from the other experts trying to use the constructs of the senior researcher. This could indicate that he has a greater variety of distinct constructs to use in such a situation that they do. In the second study the scores range from 67% to 100%. Here it is interesting to note that the trainer could use the constructs of the engineer totally, but the engineer could only use two thirds of the trainers constructs.

Except for the 100% agreement in the second study, these ratings are remarkably similar. This suggests that KSS0 shows a facility at allowing experts to compare their understanding and to determine a level of agreement. Whether the percentages are high enough to produce the required level of knowledge-base performance or too high to reach the criterion of completeness is a matter for further study.

Intra-subjective validation scores are displayed in Figure 6. These show whether the expert finds the rules meaningful, and rates the rules as correct or significant in the same way as the ENTAIL algorithm. They range in study 1 from 82% to 92%, showing a high level of expected rules appearing in the KSS output.

CONCLUSIONS AND FUTURE WORK

In KSS evaluation, internal and external contamination factors require attention. Internal factors refer to aspects of the design, its procedures, measures and observations. External factors include aspects which are external to the design but which are present in the experimental environment.

The variability of the subjects presents a consistent internal problem in evaluation studies of KSSs. The variance of the outcome measures due to the user is explicably tied to and requires isolation from the variance due to the KSS. Procedures and measures which accomplish this are required for an accurate assessment of the KSS. A measure which determines the extent that a KSS helps the user maintain consistency of input would be useful.

Developing operational definitions and formulae of reliability and validity measures require refinement and testing. The application of standard measures may be misleading and/or artificial when used with KSSs. Appropriate definitions of measurement need to be developed for KSS evaluation to be meaningful and useful.

Study 1	Rc	Rr
E1	50%	86.2%
E2	75%	89.4%
E3	83.3%	85.3%
Study 2		
E1	100%	84.4%
E2	100%	89.2%

Figure 4. Reliability Results

Expert Pairs	%
Study 1 E1's agreement with E2	76.1
E2's agreement with E1	68.2
E1's agreement with E3	72.2
E3's agreement with E1	68.8
E2's agreement with E3	72.6
E3's agreement with E2	71.1
Study 2 E1's agreement with E2	67.4
E2's agreement with E1	100

Figure 5. Inter-Subjective Validity for Ratings

Study 1	Vi
E1	88.2%
E2	91.9%
E3	82.2%

Figure 6. Intra-Subjective Validity for Rules

Evaluation studies require careful selection of subjects and selection of task demands. Subjects are likely to present different levels of knowledge and experience when faced with the same task. The effects of this difference on outcome measures are not easily determined.

The extent to which training and/or prior exposure to the KSS affects measured outcomes is also a factor in evaluation. To what extent did prior training alter the subject's approach to the KSS? This issue also suggests another: the effect due to growing familiarity with the KSS as the experiment unfolds. Subjects in the second study tended to "regress to the extreme" as they produced their grids. They produced fewer intermediate ratings at G5 than at G1 or G2. This may indicate "refinement" of the ratings or a phenomenon which occurs from the continued use of the same information for the same task demand.

Novelty effects and situational effects also require identification, elucidation and ultimately, control. What influence did the novelty of using KSS0 have on the subjects and subsequent measures? Did the amount of guidance given while the subjects worked on the system affect the measurements? Did the subjects have the right amount of time to complete the elicitations?

To what extent were the measured outputs of reliability affected due to the development of new understanding by the subject from one time to another? Lower scores for reliability may indicate a "new" or "better" understanding of the material. This development may also be attributed to the demands of the experiment.

There are many more ways in which the data is being processed, especially in terms of the rules derived from the various grids. In this paper we have looked at the similarities ane differences between experts and within experts over a 2 to 3 month time interval. Clearly, the content of these similarities and differences can be identified and related to external criteria.

One of the directions in which we are continuing this study, is to look into the objective validation of KSS. That is, to analyse the text of some research papers on this topic, using TEXAN. A comment made several times by more than one of the experts in the first study was that although this topic is taught in an undergraduate geography course, there is no textbook which covers the material in the depth that they were forced to think about it, even after the first session.

ACKNOWLEDGEMENTS

Financial assistance for this work has been made available by the National Sciences and Engineering Research Council of Canada. The KITTEN system is made available by the Centre for Person Computer Studies; the initial Apollo implementation is being carried out at the Knowledge Sciences Institute, University of Calgary. We would like to thank our colleague Brian Gaines for discussion of this work, and Robin Edmond, Karen Marion, Alvin Simms, Frank Sparks and Nigel Waters for sharing their expertise with us.

REFERENCES

Anderson, A.(1975) Policy experiments: selected analytical issues. **Soc. Methods and Research 4,1 (August).**

Bernstein, I.N. (1976). **Validity Issues in Evaluation Research.** Beverly Hills: Sage.

Burton, A.M., Shadbolt, N.R., Hedgecock, A.P. & Rugg, G. (1987).A formal evaluation of knowledge elicitation techniques for expert systems. **Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-Based Systems.** Reading, U.K.

Campbell, D.T. & Stanley, J.C. (1963) **Experimental and Quasi-Experimental Designs for Research.** Chicago:Rand McNally and Company.

Carlson, W.M. (1979). The new horizon in business information analysis. **Data Base, 10**(4), 3-9.

Chow, S.L. (1987). **Experimental Psychology: Rationale, Procedures and Issues.** Calgary: Detselig Enterprises.

Eshelman, L.,Ehret, D. McDermott,J & Tan, M (1987). MOLE: a tenacious knowledge acquisition tool. International Journal of Man-Machine Studies,Vol.26 (1),41-54

Gaines, B.R. (1987) Rapid prototyping for expert systems. Oliff, M. (Ed). **Proceedings of International Conference on Expert Systems and the Leading Edge in Productions, Planning and Control.** pp. 213-241. University of South Carolina.

Gaines, B.R. & Shaw, M.L.G. (1986). Induction of inference rules for expert systems. **Fuzzy Sets and Systems 18**(3) 315-328 (April).

Johnson, P.E. (1986a).Cognitive models of expertise. **Symposium on Expert Systems and Auditor Judgment.** University of Southern California.

Johnson, P.E. (1986b).Specification of expertise:knowledge acquisition for expert systems. **Proceedings of Knowledge Acquisition for Knowledge-Based Systems Workshop.** Banff.

Kahn, G. Nowlan, S & McDermott, J. (1985) MORE: an intelligent knowledge acquisition tool. **Proceedings of the Ninth Joint Conference on Artificial Intelligence.** Los Angeles, (August).

Kelly, G.A. (1955). **The Psychology of Personal Constructs.** New York: Norton.

Marcus, S. (1986) Taking backtracking with a grain of SALT. **Proceedings of Knowledge Acquisition for Knowledge-Based Systems Workshop.** Banff.

Michalski, R.S. (1983) A theory and methodlogy of inductive learning. In Michalski, J. Carbonell, T & Mitchell, E. (eds) **Machine Learning.** Palo Alto: Trago Publishing

Shaw, M.L.G. (1980). **On Becoming a Personal Scientist.** London: Academic Press.

Shaw, M.L.G. (1982). PLANET: some experience in creating an integrated system for repertory grid applications on a microcomputer. **International Journal of Man-Machine Studies, 17**(3), 345-360.

Shaw, M.L.G. & Gaines, B.R. (1983). A computer aid to knowledge engineering. **Proceedings of British Computer Society Conference on Expert Systems,** 263-271 (December). Cambridge.

Shaw, M.L.G. & Gaines, B.R. (1986a). Interactive elicitation of knowledge from experts. **Future Computing Systems, 1**(2) 151-190.

Shaw, M.L.G. & Gaines, B.R. (1986b). Techniques for Knowledge Acquisition and Transfer. **Proceedings of the Workshop on Knowledge Acquisition for Knowledge-Based Systems,** November, Banff, 39-0 — 39-13.

Siegel, P. (1986) **Expert Systems: A Non-Programmer's Guide to Development and Application.** TAB, Blue Ridge Summit, PA.

Sowa, J.F. (1984). **Conceptual Structures: Information Processing in Mind and Machine.** Reading, Massachusetts: Addison-Wesley.

Travers, R.M.V. (1969) **An Introduction to Educational Research (Third Edition).** MacMillan: London.

Waterman, D.A. (1986) **A Guide to Expert Systems.** Addison-Wesley: Don Mills, Ontario.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

REFINING PROBLEM-SOLVING KNOWLEDGE IN REPERTORY GRIDS USING A CONSULTATION MECHANISM

David B. Shema and John H. Boose
Knowledge-Based Systems Laboratory
Advanced Technology Center, Boeing Computer Services
P.O. Box 24346, Seattle, WA, 98124, USA

ABSTRACT

A general problem when modifying knowledge bases is that changes may degrade system performance. This is especially a problem when the knowledge base is large; it may be unclear how changing one item in a knowledge base containing thousands of items will affect overall system performance. Aquinas' knowledge representation techniques and consultation review mechanisms help alleviate this problem. The consultation review mechanisms are discussed here.

We are experimenting with ways to use consultations and test cases to refine the information in an Aquinas knowledge base. The domain expert can use interactive graphics to specify the expected results. Modifications to the knowledge base may be tested against previous consultations; adjustments are suggested that make the results of all previous consultations as well as the current consultation correlate better with the expert's expectations. New traits are synthesized that would improve the performance of all previous consultations. New test cases are suggested that cover aspects missed by previous test cases. While we are just beginning to experiment, these techniques promise to provide help in improving problem-solving performance and gaining problem-solving insight.

INTRODUCTION

Before discussing the consultation mechanisms, a brief description of Aquinas and its reasoning methods will be given.

Description of Aquinas

Aquinas is being developed at Boeing Computer Service's Advanced Technology Center (Boose, 1986; Boose & Bradshaw, 1987; Boose, 1987). The goal of Aquinas is to gather problem-solving information for most analytic types of problems from a domain expert with minimal outside guidance.

Aquinas is a knowledge acquisition workbench that gathers knowledge from one or more experts and stores it in a hierarchical network of repertory grids. This knowledge can be examined and refined using tools that do clustering, similarity analysis, implication analysis, and consultation testing. These tools use techniques to analyze the information in grids and suggest ways to refine the knowledge base.

Consultations serve as a way of validating the current knowledge base against real world test cases. Aquinas permits an expert to check the knowledge against a set of test cases. Contrasting the results of consultations with the expert's expectations provides a way of

reviewing and revising the contents of a knowledge base. Trend analysis can measure the progress that is made as new knowledge is acquired and tested.

Personal Construct Psychology and Repertory Grids

Interviewing methods from Personal Construct Psychology are used to elicit an expert's knowledge which is captured in hierarchically organized *repertory grids* (also referred to as *rating grids*). During an initial interview the expert is asked to specify a case or a set of cases defining a problem where this knowledge is useful. Next, he is asked to supply a list of possible solutions to the problem. Traits are then elicited to distinguish between the possible solutions. A repertory grid of traits and solutions is created. The expert rates each solution against each trait and supplies a weighting factor for each of the traits. Some traits may more heavily influence a decision than others. A repertory grid is shown in Figure 1.

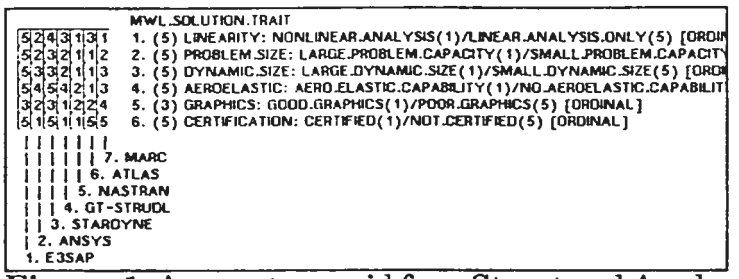


Figure 1. A repertory grid for a Structural Analysis Advisor.

When the first phase of interviewing is complete, an initial knowledge base exists. This knowledge base is then expanded and refined. Solutions and traits are added and deleted. Specific traits or solutions may be generalized or broken down into more specific items with the addition of nested repertory grids. The existing knowledge can be examined for correctness and consistency. Additional experts can develop their own sets of solutions, traits, and ratings. The knowledge acquired from multiple experts can be used in consultations to produce consensus and dissenting opinions.

CONSULTATIONS IN AQUINAS

Consultation Mechanism

After a grid is constructed a *consultation* can be run. The expert identifies the subset of the knowledge base to be used, provides a description of the consultation context, chooses a reasoning method for the consultation, and assigns a preference or observation rating for each (Figure 2). The expert's certainty for each preference or observation may be distributed between multiple rating values.

is used to pre-test performance when using Aquinas to make a rule-based knowledge base for another expert system shell (S.1, M.1, KEE, OPS5, and so on).

Maximum entropy. A maximum entropy approach (Shastri and Feldman, 1985; Cheeseman, 1983) may be used to propagate evidence through hierarchies of grids. A cost / benefit analysis of traits may be performed. The measure of information is similar to that used by Quinlan, where questions that yield the most distinguishing information are asked first (Quinlan, 1983). Expertise from several experts may be weighted and combined.

Analytic Hierarchy Process. The Analytic Hierarchy Process is a method of reasoning about preference tradeoffs in a decision. It orders the set of possible solutions by mapping grid ratings onto judgment matrices (Saaty, 1980). A principal eigenvector is computed for each matrix, then the eigenvectors at different levels in the hierarchies are combined to yield a final ranking.

CONSULTATION REVIEW MECHANISMS

Consultation review mechanisms in Aquinas are discussed in this section.

Changes that lead to knowledge base performance improvement (or degradation) take several forms:

Changing ratings in a grid. Judgments about the relationships between solutions and traits may be revised.
Changing a trait's weight. A trait's influence may be increased or decreased by changing its weight.
Adding or deleting a trait. Traits that affect the outcome may be added, deleted, or moved. New traits are rated against existing solutions and given weights.
Adding or deleting a solution. The potential set of solutions that affect the outcome may be expanded or restricted. New solutions are rated against existing traits.
Changing the expectations of outcomes. The expert may revise the expectation of the system's behavior.
Re-organizing existing knowledge. Knowledge can be reorganized to change its affect on performance. For instance, knowledge can be regrouped under different cases in the knowledge bases hierarchies so that it is applied under different circumstances.

Review mechanisms fall into several categories - *manual*, *directed*, *semi-automated*, and *automated*.

Manual Methods - Inspect Results and Revise

A variety of methods exist to manually inspect consultation results. The expert may review the results and change the knowledge base directly to try and improve the system's performance.

The rank ordering of the solutions may be displayed in list form (as above), tables, bar charts, or pie charts. Tracing of an ongoing consultation can be done graphically using bar charts to display the current stage of the ranking, or tracing information can be displayed as text. The amount of tracing can be controlled by the expert (Figures 4, 5, and 6).

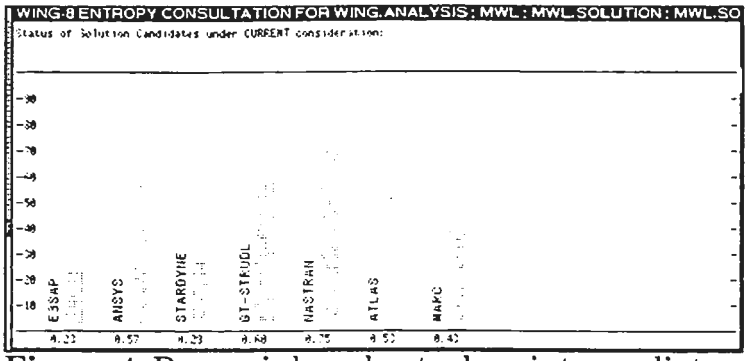


Figure 4. Dynamic bar charts show intermediate consultation results. The effects of each trait can be seen on the solution set. After watching the effect of a particular trait, the expert may manually adjust the rating grid information to decrease or increase the effect of the trait on the solution set.

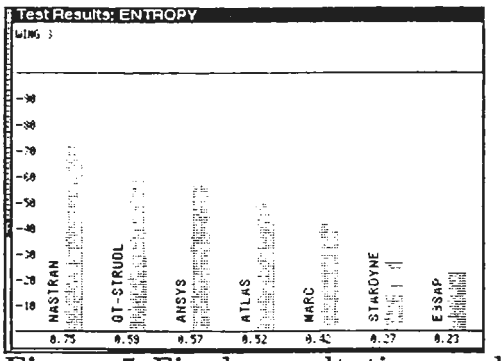


Figure 5. Final consultation results may be displayed as a bar chart. This graphic representation helps the expert see the *intervals* between solutions, while lists help the expert see the *overall ordering*.

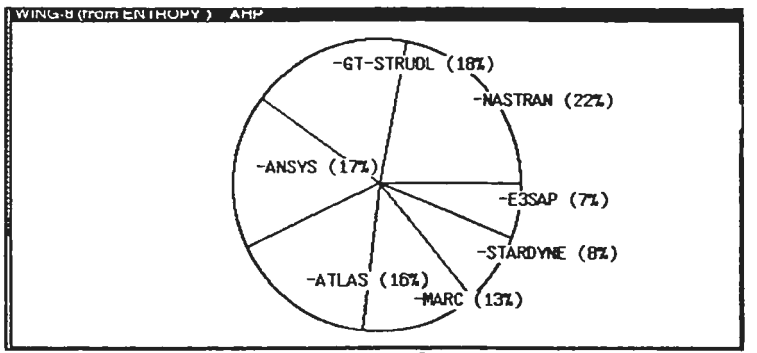


Figure 6. Consultation results can also be displayed in normalized form in a pie chart. It is often easier to see the *relative weight* of each solution in a pie chart (as opposed to the *absolute weights* emphasized in a bar chart).

Each individual rating's contribution to the solution may be emphasized by highlighting those grid elements which come closest to preference ratings given during a consultation (Figure 7).

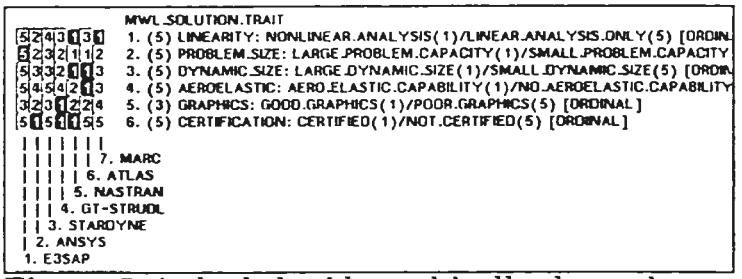


Figure 7. A shaded grid graphically shows the contribution of each rating in a particular consultation. Black (inverted) ratings contributed the most; white ratings contributed the least. The expert can quickly see the information that contributed to the particular test case results.

All of these lists, graphs, charts, and grids may be saved from consultation to consultation. The expert can compare previous results to current results and see if changes led to improvements. These displays can also be recreated from saved consultations.

Directed Methods - Elicit Results Expectations, Score Results, Revise

The correlation between the expert's expectation of the solution ranking vs. the outcome of a consultation may be calculated. This correlation is used as a measure of the quality of the knowledge base performance for a case. Solutions that are out of place can be used to elicit new information.

Eliciting expectations. The expert may partially or completely specify the expected outcome of a consultation.

Specifying those solutions that should be at or near the top and bottom of the ranking may be sufficient:

```
Do you wish to specify your expected ordering at this time?
AQU** YES

Select those solutions that are at the top of the ordering, one
per line. Enter a carriage return when finished.
TOP**  NASTRAN
TOP**  (CR)

Select those solutions that are near the top of the ordering,
one per line. Enter a carriage return when finished.
NEAR.TOP**  ANSYS
NEAR.TOP**  (CR)
```

```
Select those solutions that are near the bottom of the
ordering, one per line. Enter a carriage return when finished.
NEAR.BOTTOM**  MARC
NEAR.BOTTOM**  (CR)

Select those solutions that are at the very bottom of the
ordering, one per line. Enter a carriage return when finished.
BOTTOM**  E3SAP
BOTTOM**  (CR)
```

Expectations may also be specified completely:

```
Would you like to compare the way the results were ranked with
the way you would rank the results?
AQU** YES

How would you rank the solutions (list them in order, one to a
line)?
AQU** NASTRAN
AQU** ANSYS
AQU** ATLAS
AQU** GT-STRUDL
AQU** MARC
AQU** STARDYNE
AQU** E3SAP
```

Scoring results. Results are scored against expectations by computing a rank correlation measure:

```
The correlation  0.86  between your expectations  and the actual
results is  significant at the .05 level.
```

Correlation scores show the expert how well the knowledge base performed, given the expectations. Correlation scores recorded over time for many test cases can show whether or not the knowledge base is improving as revisions are made.

New Triads Based on Expectations. Aquinas calculates the solution that is most out of place and asks the expert for a new trait based on where the solution *should* have appeared in the list. When this new trait is included in a subsequent consultation the out-of-place solution should be closer to the expected place in the list (Boose, 1986):

```
The solution GT-STRUDL is most out of place.

What trait do GT-STRUDL and ATLAS share that make them different from
ANSYS?
NEW TRAIT (EXTREME)**
```

Semi-Automated Methods - Suggesting Changes

Solution results adjustment. Implementation of these methods is in progress. If the expert disagrees with the rank ordering of the solutions, he will be able to modify the bar chart displaying the results of the consultation. His new ordering will be used to find potential changes to ratings or trait weights that would lead to the new ranking. Changes to *rating grid values* may be suggested that would help push target solutions up or down on the results list. Changes to *trait weights* may be suggested that would help push groups of solutions up or down on the results list. After information, is modified the consultation will be replayed and the new and old results will be compared. This method of interaction is similar to a spread-sheet: the expert modifies the answer graphically and the data is used to calculate changes.

Automated Methods - Rating and Trait Weight Revision, Synthesizing New Traits

Preliminary results are presented for methods that examine the knowledge in a rating grid and automatically generate changes that lead to performance improvements.

Automatic Rating and Weight Adjustment. Values of ratings and trait weights may be varied automatically to produce a new rating grid. Batch test cases are run, and the results of the changes are measured against the expert's expectations. A new change is selected, and the test cases are re-run and scored.

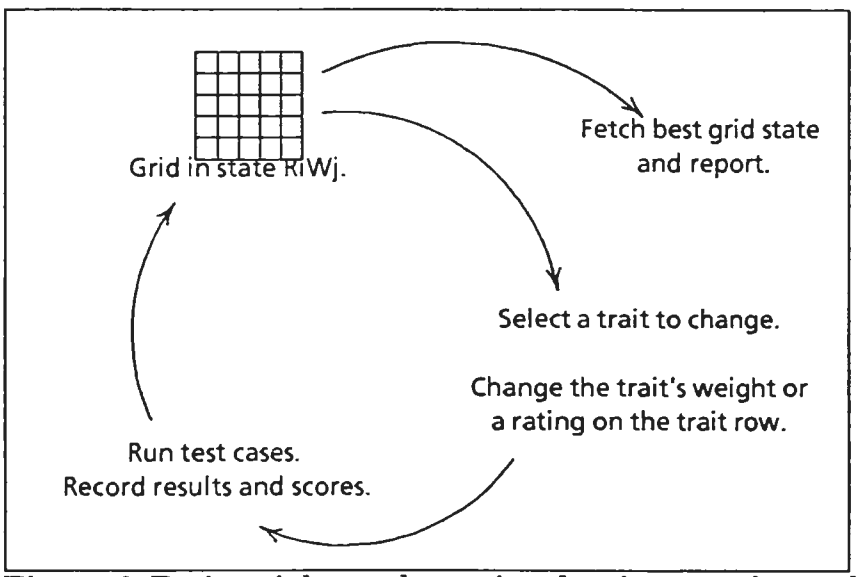


Figure 8. Trait weights and associated ratings are changed automatically, batch test cases are run, and the results are scored against the expert's expectations for those cases and stored. Heuristics are used to select the next trait-weight-rating change. When finished, the best combination of trait weights and ratings is reported to the expert. "Best" is defined as the highest average correlation score over the test case set for that grid state. The expert can examine the proposed grid and direct Aquinas to implement all or a portion of the suggested changes.

Figure 8 shows the grid modification, test, score, and report cycle. The expert starts by specifying a set of test cases and expected outcomes (a rank-ordered solution set for each test case).

Since exhaustively checking each possible combination of ratings and weights is not feasible, heuristics are used to select a new change and decide when to stop work. Heuristics are based on:

the original trait weight (traits with higher weights tend to be tried sooner),
the original rating distribution spread in the row,
the distance from the original grid state to the proposed new grid state (closer states are tried first since it is assumed that the grid from the expert is reasonably close to a final state), and
previous results (if moving a rating or weight in a certain direction produces better results, try pushing it farther in that direction).

When the process is over, several results are reported:

The grid that exhibited the best performance. The expert can examine the new grid to see whether or not the changes make sense. If the expert agrees, the changes are automatically implemented in the existing grid.

A sensitivity ranking for the traits. The overall effect each trait has on all the tested combinations is reported. Traits are ranked according to the greatest performance variance and reported. This information can help focus the expert's attention on traits that most affect performance.

A "completeness" score for the test case results. A score is given that shows the overall completeness of possible test case results rankings for the best (or any other) grid (for instance, there would be twenty-five possible outcomes for five possible solutions). This helps the expert see how well the test cases cover the set of solutions. This information can be used to generate new potential test cases (see below).

We have only tried this technique on a few problems, but we have seen promising results. Minor sets of changes have yielded significant problem-solving improvement over small sets of test cases.

Analytic and Synthetic Changes. Using Aquinas, the best case of knowledge base modification (from a maintenance point of view) is when adjustments made solely to trait values and weights result in improvements of (or at lease, do not degrade) all existing cases. These are referred to as *analytic* changes, since no new traits or solutions are added, and a change can be made by analyzing relationships between existing traits and solutions. Usually this is not the case; a given change may improve some case performance and degrade other case performance. When this happens, knowledge must be added and/or deleted. Adding new knowledge is referred to as a *synthetic* change.

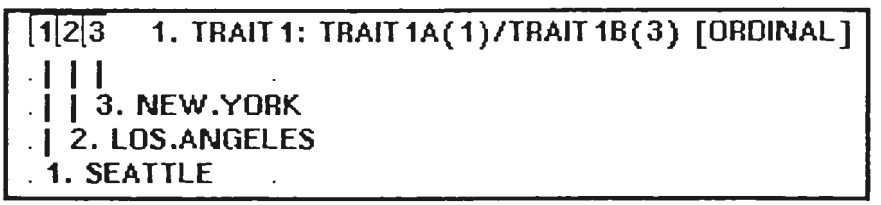
Synthesizing New Traits. Examination of the consultations makes it possible to hypothesize new traits that would continue to keep the knowledge base correct for existing cases. A new trait is formed internally for a set of solutions; its values are systematically varied and combined with existing traits and tested against existing consultations. If the inclusion of this trait improves the overall scores of existing consultations, the expert is

asked to try and name it, and it is included in the knowledge base. Such traits can also be used to start a *new* grid.

For example, suppose the expert supplies test case results from past experience. The results are in terms of rank-ordered lists of solutions. The expert supplies Test Case 1, which looks like this:

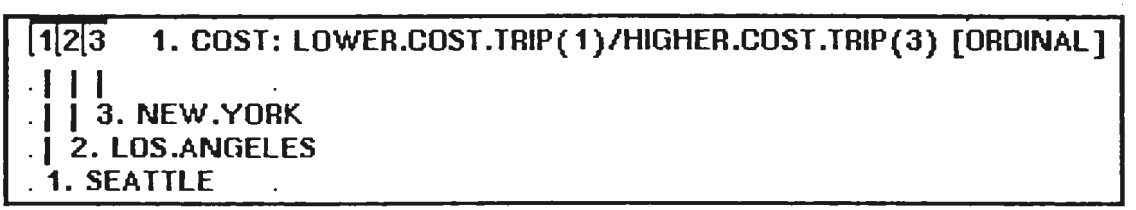
Test Case 1:
Seattle
Los.Angeles
New.York

In this case, Seattle won, Los Angeles came in second, and New York came in last. Aquinas could automatically construct a rating grid with one trait that could produce these results during a consultation, even if we don't know the name of the trait:



Aquinas would fill in ratings by ordering them in the same way as the test case. Assuming that the trait is ordinal and that its rating scale is from 1 to N, where N is the number of solutions, Seattle is rated 1 (it came in first in the test case), Los Angeles is rated 2 (it came in second), and New York is rated 3 (it came in last). By selecting a preference of "1" for TRAIT1 during a consultation, we would get the desired test case results in order.

Aquinas would make up a temporary name for the trait - TRAIT1 with extremes TRAIT1A and TRAIT1B. Aquinas would then ask the expert to identify an important trait that would produce these ratings for these solutions. The expert realizes that a trait COST will extremes LOWER.TRIP.COST and HIGHER.TRIP.COST could correspond to this unnamed trait:



So, given the test case supplied by the expert, Aquinas could synthesize a new trait complete with rating values, and the expert could name the trait. Now suppose that the expert adds a second test case involving the same solutions, but with different results:

Test Case 2:
Los.Angeles
New.York
Seattle

Aquinas could add a new trait and ratings to the rating grid as follows:

```
|1|2|3   1. COST: LOWER.COST.TRIP(1)/HIGHER.COST.TRIP(3) [ORDINAL]
|3|1|2   2. TRAIT2: TRAIT2A(1)/TRAIT2B(3) [ORDINAL]
. | | |
. | | 3. NEW.YORK
. | 2. LOS.ANGELES
. 1. SEATTLE
```

If we were to run a consultation and give "unknown" as a preference for COST and "1" as a preference for TRAIT2, we will get the desired results for Test Case 2 (if a trait preference is "unknown" then that trait is ignored during the consultation). On the other hand, if we run an additional consultation and give "unknown" as a preference for TRAIT2 and "1" as a preference for COST, we will still get the desired results for Test Case 1.

Again, Aquinas would ask the expert to name the trait:

```
|1|2|3   1. COST: LOWER.COST.TRIP(1)/HIGHER.COST.TRIP(3) [ORDINAL]
|3|1|2   2. ACTIVITIES: MORE.THINGS.TO.DO(1)/FEWER.THINGS.TO.DO(3) [ORDINAL]
. | | |
. | | 3. NEW.YORK
. | 2. LOS.ANGELES
. 1. SEATTLE
```

Any number of new traits could be added, rated, and named in this way, one trait-per-test case. The method assumes that 1) there exists at least one trait that completely rank-orders (dominates) the solution set in the proper way (this may not be true when complex trait interactions are needed to produce the desired results), 2) the expert can name the trait (this becomes increasingly difficult as the solution set size increases), and 3) rank-ordered test cases can be created or are already available. These limitations probably restrict the method to small problems, but it could be an interesting, insightful way to help the expert construct new grids that automatically exhibit the proper behavior.

Traits could also be combined together when their ratings are "opposite" each other (that is, when the test case solution sets that produced the grids are in opposite rank orders). Suppose the expert added a third test case:

Test Case 3:
New.York
Los.Angeles
Seattle

Aquinas would generate a new trait, and the expert would name it:

```
1 2 3   1. COST: LOWER.COST.TRIP(1)/HIGHER.COST.TRIP(3) [ORDINAL]
3 1 2   2. ACTIVITIES: MORE.THINGS.TO.DO(1)/FEWER.THINGS.TO.DO(3) [ORDINAL]
3 2 1   3. DISTANCE: FARTHER(1)/NEARER(3) [ORDINAL]
. | | |
. | | 3. NEW.YORK
. | 2. LOS.ANGELES
. 1. SEATTLE
```

Traits 1 and 3 could be automatically combined as follows while not adversely affecting the system's behavior:

```
1 2 3   1. COST-DISTANCE: LOWER.COST.TRIP-NEARER(1)/HIGHER.COST.TRIP-FARTHER(3)
3 1 2   2. ACTIVITIES: MORE.THINGS.TO.DO(1)/FEWER.THINGS.TO.DO(3) [ORDINAL]
. | | |
. | | 3. NEW.YORK
. | 2. LOS.ANGELES
. 1. SEATTLE
```

The expert could then choose to retain the new combined trait or rename it as a new trait that included both aspects.

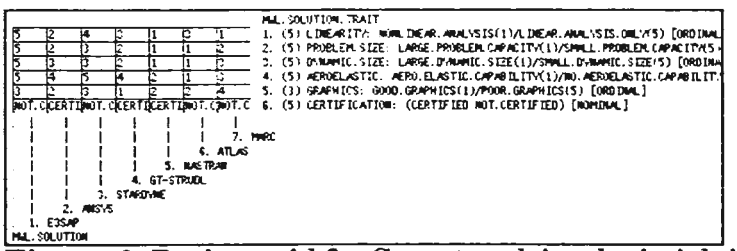


Figure 9. Rating grid for Structural Analysis Advisor. Aquinas will synthesize a new trait for this grid based on expected consultation results.

Traits may also be synthesized as a result of examining the expert's results expectation list. Below, the expert entered an expectation list after a consultation based on the grid in Figure 9:

```
How would you rank the solutions (list them in order, one to a
line)?
AQU** NASTRAN
AQU** ATLAS
AQU** ANSYS
AQU** MARC
AQU** GT-STRUDL
AQU** STARDYNE
AQU** E3SAP
```

Aquinas synthesizes a new trait based on the expectation list. It is an ordinal trait with a scale range from 1 to 7 (Figure 10).

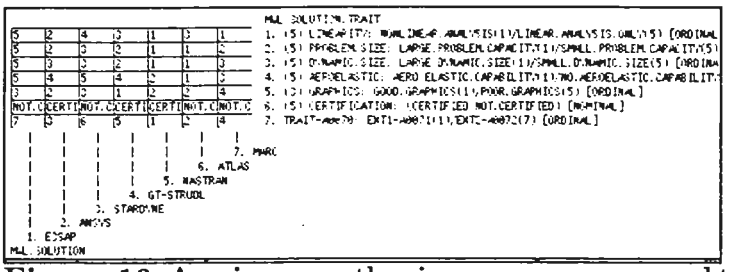


Figure 10. Aquinas synthesizes a new, unnamed trait based on the expert's expectation list.

After examining the values, the expert decides that the proposed trait would correspond to a new trait, LIBRARY.SIZE, with extremes LARGE.ELEMENT.LIBRARY and SMALL.ELEMENT.LIBRARY. Inclusion of this new trait improves the grid's problem-solving behavior for that consultation test case.

Implementation of the following two methods is in progress.

Consultation synthesis. Examining the contents of previous consultations may suggest new test cases that are significantly different from those already present (that is, their rank orders are significantly different from any existing test cases). The expert could evaluate these synthesized test cases and try to see if they fit any real-world situations. The knowledge base validation effort could be assisted by the expanded coverage of these test cases.

Test Case Analysis. Comparisons of consultation results might highlight discrepancies in the expected results that the expert has given. Test cases may not be appropriate for the section of the knowledge base under consideration, or the expert may have made an error in his ranking. Validation of the test cases often is as important as the validation of the knowledge base.

3.0 SUMMARY

The Aquinas workbench contains a number of tools to assist a domain expert intent on capturing his problem-solving knowledge in a knowledge base. Tools for analyzing consultations to improve performance are an important part of the workbench. These tools may modify the knowledge base after checking that existing test cases still give the intended results. Synthesis of traits and test cases help with the validation of the knowledge base. Experiments are underway to measure the utility of these tools.

4.0 REFERENCES

Adams, J., Probabilistic Reasoning and Certainty Factors, in B. Buchanan and E. Shortliffe (Eds.), Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Reading, MA: Addison-Wesley, 1985.

Boose, J. H., A Knowledge Acquisition Program for Expert Systems Based On Personal Construct Psychology, *International Journal of Man-Machine Studies*, Vol. 23, 1985.

Boose, J. H., Expertise Transfer for Expert System Design, New York: Elsevier, 1986a.

Boose, J. H., and Bradshaw, J. M., Expertise Transfer and Complex Problems: Using Aquinas as a Knowledge Acquisition Workbench for Expert Systems, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1, 1987.

Boose, J. H., Use of Repertory Grid-Centered Knowledge Acquisition Tools for Knowledge-Based Systems, *Proceedings of the 2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, 1987.

Bradshaw, J. M., and Boose, J. H., Decision Analysis Techniques for Knowledge Acquisition: Combining Information and Preferences with Aquinas, *Proceedings of the 2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, 1987.

Cheesman, P., A Method of Computing Generalized Bayesian Probability Values for Expert Systems, *Proceedings of the Eighth IJCAI*, Karlsrue, Germany, August, 1983.

Hayes-Roth, F., Waterman, D.A., Lenat, D.B., Building Expert Systems, Addison Wesley, 1983.

Lenat, D.B., Teiresias: Applications of Meta-Level Knowledge, in Davis, R., and Lenat, D. B., Knowledge-Based Systems in Artificial Intelligence, McGraw-Hill 1982.

Quinlan, J. R., Learning Efficient Classification Procedures and their Application to Chess End Games, in Michalski, R. S., Carbonell, J.G., and Mitchel, T. M. (Eds.), Machine Learning - An Artificial Intelligence Approach, Vol 1, Tioga: Palo Alto, CA, 1983.

Saaty, T.L., The Analytic Hierarchy Process, McGraw-Hill, 1980.

Shastri. L., and Feldman, J., Evidential Reasoning in Semantic Networks: A Formal Theory, in the *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August, 1985.

AN EXPLANATION BASED APPROACH TO KNOWLEDGE-BASE ACQUISITION

Kenneth Silvestro, Ph.D.
Computer Science and
Information Studies
Adams Hall
Bowdoin College
Brunswick, Maine 04011

ABSTRACT

An expert system capable of "intelligent behavior" requires access to a large store of knowledge called a knowledge-base. A practical approach to knowledge-base evolution, addressing the complex task of hand-crafting domain specific knowledge, has become necessary. One solution is computer acquisition of knowledge. User supplied natural language explanations provide our artificial intelligence acquisition tool with the necessary information to construct and develop a domain specific knowledge-base for a target expert system. The theory of acquisition is founded upon a multi-process strategy relating individual utterance "types" to a unique acquisition process and the underlying structural elements of explanations.

INTRODUCTION

Expert systems capable of "intelligent behavior" require access to large amounts of knowledge stored in repositories called knowledge-bases (KBs). The knowledge within the KBs is a symbolic representation of facts, entities and rules pertaining to a particular domain. Constructing and extending such KBs has proven to be an extremely time consuming and difficult task, since the researcher must design and structure each symbolic representation individually.

This research effort focuses upon a practical approach to the problem of knowledge-base evolution. Systems capable of automatically constructing and extending KBs provide a particularly desirable resolution to the problem. In addition, an acquisition system which is user-friendly would be of greater benefit. KBAM (Knowledge-Base Acquisition Mechanism) is an expert system tool which constructs KBs by accepting natural language (i.e., English) explanations from the user.

The user provides KBAM with explanatory text describing the critical domain knowledge required for an expert system. Each natural language utterance within the explanation is first processed by a conceptual analyzer which is an integral component to the acquisition processes. The semantic representation of each utterance produced by the analyzer is then processed by KBAM during the construction of a KB.

The development of the target KB is based upon a multi-acquisition process which recognizes individual utterances within

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

explanations. Utterance recognition is accomplished by classifying utterances within predesigned categories; that is, the type of utterance and the structural stage of the explanation to which the utterance is a member. Thus far, a small number of utterance types have proved sufficient for describing domain knowledge and three structural stages have been uncovered.

KNOWLEDGE ACQUISITION FROM EXPLANATION

Underlying many modes of learning (e.g., being told, by analogy, etc.) is a common factor - explanation. One must remember that explanations imply intention and can vary in their "mode" of presentation:

1. Explanations can be expressed in natural language.

2. Explanations may also be expressed in actions or directed movements.

3. A picture can serve as an explanation.

4. Discovery or unexpected events can also serve as explanatory; although, of course, there is no sentient "explainer."

Explanation is a central component of human communication processes. We depend upon explanations to communicate and achieve an understanding of the world around us. As an example, consider the explanation shown below which describes the construction of a desk.

> The first thing to do is to construct the dust panels. For those of you not familiar with the term dust panel, they are the drawer supports. You construct them by cutting a rectangular frame to the correct dimension. Each inner side of the frame must have a groove or slot cut along the middle. Into these slots insert and glue a piece of plywood. The number of dust panels needed depends upon the number of drawers. Now that the dust panels are complete, the pedestals can be constructed. The pedestals are the parts of the desk that are on either side of the leg space, the areas where the drawers are found. To construct the pedestals, grooves must be cut into the sides of the pedestals (the inner side of the pedestals, the part not exposed). These grooves must be large enough to fit the dust panels and should be separated from each other by the height dimension of the drawers. Insert and glue the dust panels into the grooves of the pedestal sides. The pedestals are now complete and should be firm and stable. Finally (but certainly not least) cut the desk top to size. Place it on the top of the pedestals and permanently fasten it to the pedestals.

Communicated through this explanation is the process of constructing the various desk components followed by the construction of the desk itself. Assuming the reader had no understanding of desk construction, presumably what has been acquired is a rudimentary understanding of this process. This is true only if each individual utterance within the overall explanation and its contextual relation is understood.

Explanations such as this one are so commonplace that we seldom have a need to analyze their contents in any depth. Often explanations are processed (understood and assimilated) for their knowledge content without regard to the structure of the explanation or how the individual utterances transfer knowledge. In the construction of an acquisition tool, based upon natural language explanations, such structural distinctions must be considered.

The above explanation has a number of structural elements. Initially we encounter explanatory utterances (i.e., each individual utterance within the explanation) which function as "defining" utterances: for example, "... the term dust panels, they are drawer supports." Utterances which tend to define the function of, or the essence of, have been typed definitional, but more importantly, belong to a phase within the explanation which defines domain primitives. Next we encounter explanatory utterances describing "how to" ideas. These ideas are related to the "subject" of the definitional utterances. For example, how to construct or how to use the newly acquired conceptual entities (e.g., dust panels). This, in turn, is followed by further explanatory utterances which combine "defining" with "how to" utterances. These eventually lead to a "joining of the parts." That is, a description which combines the previously defined and functional characteristics of the individual parts into a whole unit - the focus behind the entire explanation. Briefly summarizing, the following structural elements have been described:

1. Defining or functional utterances - the FOUNDATIONAL stage.

2. How to use or develop the newly acquired information- the CONSTRUCTIVE stage.

3. Combining the explanatory utterances into a whole (i.e., an explanation) - the CONJOINT stage; where the goal of the explanation is made clear or can be simply inferred.

The above categorical stages and associated explanatory utterances are common characteristics of explanations.

To date, nine different explanatory utterances have been typed and utilized within the explanations presented to KBAM. Each type is differentiated by one or a combination of the following characteristics:

1. The conceptual classification of a meaning representation.

2. The overall relationship between imbedded conceptual constructs found at the meaning level.

3. Specific characteristics found within the meaning representation.

Recall that each explanatory utterance is first processed by a conceptual analyzer producing a semantic representation for KBAM. KBAM then differentiates the appropriate stage and utterance type, activating the corresponding acquisition process.

The nine types are briefly described below.

1. DEFINITIONAL - an explanatory utterance introducing an unknown concept based upon a known concept. Found within the foundational stage of the explanation.
   EXAMPLE: A STUDENT IS A PERSON,

   where STUDENT is unknown but PERSON is known.

2. ENTITY-TAGGING - explanatory utterance which names entities. Found within the constructive stage of the explanation.
   EXAMPLE: COMPUTER-SCIENCE IS THE LABEL OF A DEPARTMENT.

3. EXTERNAL-ATTRIBUTE - an utterance describing a new characteristic (or an attribute) of a concept. Found within the constructive stage of the explanation.
   EXAMPLE: A NUMBER IS PART OF THE LABEL OF A COURSE,

   where the identifier for an academic course is being explained. It consists of a number and a department (e.g., CS 321).

4. CAUSAL - used with causal relations. Found within the constructive stage of the explanation.
   EXAMPLE: IF A STUDENT PASSES CS 130 THEN THE STUDENT ACHIEVES 4 CREDITS.

5. EPISODE-ENTITY - explanatory utterance describing participants of episodes. Found within the constructive stage of the explanation.
   EXAMPLE: A STUDENT PLAYS A ROLE IN A COURSE.

6. GOAL-SPECIFIER - utterances describing the goals of episodic participants. Found within the constructive stage of the explanation.
   EXAMPLE: IF A STUDENT WANTS TO LEARN INTRODUCTORY PROGRAMMING THEN A STUDENT MUST TAKE CS 110.

7. EVENT-CHAINING - utterances which present events or episodes "chained in time." Found within the constructive stage of the explanation.
   EXAMPLE: CS 110 IS THE PREREQUISITE FOR CS 111.

8. FACTUAL - a default.
   EXAMPLE: CS 130 IS DIFFICULT.

9. CONCEPT-RELATION - utterances describing an action concept. Actually, action concepts relate to verb-type words. Found within the foundational stage of the explanation.
   EXAMPLE: AN INSTRUCTOR TEACH A STUDENT MEANS AN INSTRUCTOR TRANSFERS INFORMATION TO A STUDENT,

   where TEACH is the concept being acquired.

Together, the typing of the explanatory utterances and the structural stages within explanations form the basic theory behind the acquisition processes. As such, a practical tool for acquiring a KB for an expert system has been developed.

OVERVIEW OF KBAM

Until recently, natural language processing had been relatively unsophisticated and little was known of knowledge acquisition. The current day sophistication achieved by natural language processors enables expert system tools, such as KBAM, to explore an entirely new realm of "human-like" processing.

KBAM is a rule-based implementation of one theory of acquisition - acquisition by explanation. Its acquisition processes function at a "conceptual" level (Schank, 1973), permitting KBAM to be extremely flexible with respect to natural language input and domain. A block diagram of KBAM depicting the acquisition process is shown in Fig. 1.

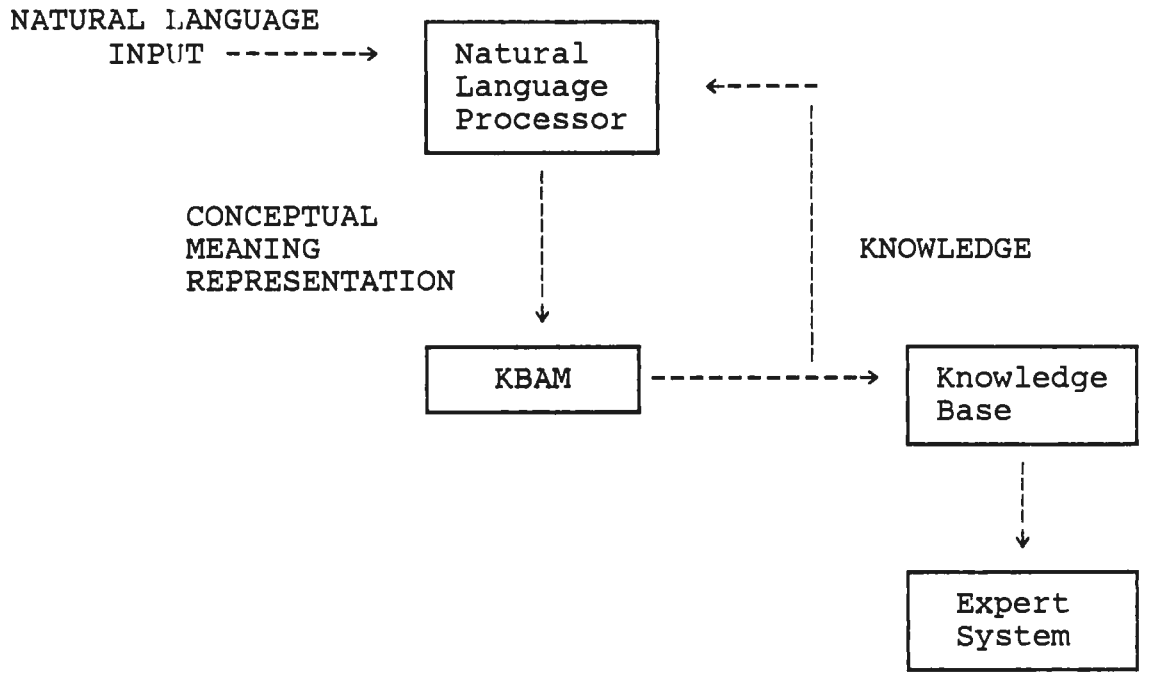


Fig. 1: Flow Diagram Depicting Acquisition Process

NLP and KBAM

The natural language processor (NLP), or the conceptual analyzer (Birnbaum, 1979; Riesbeck, 1975), forms the conceptual meaning representations - the semantic representations. As depicted within Fig. 1, KBAM reciprocates by supplying the NLP with new lexicon entries. In this manner, a rudimentary form of language acquisition is also being accomplished. The NLP is then capable of processing explanations which prior to specific language acquisitions would be impossible.

Although intimately connected, one way to differentiate the two components requires categorizing the primitive knowledge found within each. As one might expect, NLP's knowledge can be categorized as world knowledge pertaining to natural language understanding, while KBAM's knowledge can be categorized as world knowledge pertaining to the acquisition process. This in itself is not very illuminating, however, it does provide a reference point. NLP's lexicon includes many words of a general nature, for example, function words (a, the), entities (episode, person) and action concepts (give, want). Such words are domain independent, and therefore, classified primitive world knowledge. KBAM, on the other hand, includes primitive world knowledge assertions related to the primitive words (or world knowledge concepts) found within NLP's lexicon. That is, KBAM possesses primitive assertions associated with the NLP concepts. The primitive world knowledge within the two components leads to the following conclusions:

1. Since KBAM has primitive assertions associated with NLP's primitive concepts, we must consider NLP's concepts part of KBAM's primitive knowledge.

2. KBAM has the potential capacity of acquiring knowledge for a number of mutually exclusive domains.

Claim #2 follows from the fact that primitive concepts within NLP's lexicon and the associated assertions found within KBAM are domain independent. As an example, consider multi-sense words. Whenever a particular word sense unique to a domain is encountered, it must be acquired by KBAM. Since it is domain specific, it would not be a primitive concept found within NLP's lexicon.

Claim #1 integrates rather than differentiates NLP and KBAM. This is due to the fact that NLP and KBAM are "sharing" the same long-term memory - NLP's lexicon. Although the knowledge is separated into two discrete components, we cannot avoid perceiving it as one long-term memory.

Architectural Structure

KBAM's architecture is a production system (Newell, 1973), where the productions are characterized as TASKS and RULES. The major advantage of this architecture is the natural movement toward an incremental development of the acquisition tool.

TASKS and RULES are data structures which embody both static and procedural knowledge. Structurally and in the manner in which they are activated, they are identical. Their differences lie in their functionality. TASKS control the flow or direction of the acquisition process while RULES perform modifications to knowledge structures or other data structures found within KBAM's working memory. Because of their functional differences, TASKS are considered high-level controllers and RULES are considered low-level modifiers when viewed from a hierarchical standpoint.

In complex production systems which have many productions, it is not possible to create orthogonal conditions (unique start tests) without a great deal of effort (McDermott, 1978). As new productions are given to the system, unique conditionals must be created by testing against existing productions. These conditionals have two destinies, either they become extremely complex or failure to produce orthogonality is the end result (production conflicts result). In order to resolve such conflicts and avoid the extra effort of creating complex conditionals, KBAM has been designed with its own attention focusing mechanism.

The attention focusing mechanism is based upon the existence of many high-level mechanisms (CONTROL UNITS); their purpose being to select a subset of RULES and TASKS for execution. Each CONTROL UNIT becomes active only under particular processing needs (e.g., in recognizing an utterance type). Once activated they control the dissemination of information and the activation

of the lower-level mechanisms (PERFORMANCE UNITS) found at KBAM's working memory level. CONTROL UNITS are implemented as clusters of TASKS and PERFORMANCE UNITS as clusters of RULES. The highest level TASKS receive input directly from the NLP mechanism. They then decide which CONTROL UNIT is needed to perform the acquisition process of the input information. The selected CONTROL UNIT then assumes the control responsibility for the acquisition process.

Adding new acquisition processes requires clusters of new TASKS (CONTROL UNITS) to be included within the production system; they function at the top-level to focus KBAM's attention to new or old PERFORMANCE UNITS. If a new PERFORMANCE UNIT is to be added, the RULE conditionals must be unique, but only within this new cluster. Each cluster is independent and only becomes active via a CONTROL UNIT. There is a tremendous reduction in the number of production tests which must be orthogonalized. Orthogonalization must be applied locally, within CONTROL UNITS and PERFORMANCE UNITS and with the top-level TASKS of each CONTROL UNIT.

Structural Stages and Utterance Types

The current implementation of KBAM utilizes the structural stages described above in the simplest manner - to "guide" the utterance recognition process. Each structural stage is recognized by a unique TASK which in turn activates other TASKS enabling the differentiation of the many associated utterance types within a structural stage. Collectively, the TASKS used to recognize a particular structural stage and an associated explanatory utterance form a top-level CONTROL UNIT. As stated above, the CONTROL UNITS direct the flow of the acquisition process through the hierarchical architecture comprised of further CONTROL UNITS and PERFORMANCE UNITS.

The activated structural stage can be very useful beyond merely recognizing different utterance types. For example, upon entering a particular structural stage, expectations related to this stage could be activated for potential use. KBAM currently has the capability to activate production rules (a RULE or TASK) for potential use; however, specific expectations associated with a stage and acquisition have not been implemented. In an interactive system, the "effort" required by the system to formulate questions could be greatly reduced if the question formation is accomplished within the limitations of a structural stage. If, for example, within the constructive stage a question was composed, it would pertain to the "how to" aspect of a domain primitive acquired during the foundational stage. The intentionality behind the question becomes more manageable from a system design consideration. In more sophisticated systems where inferencing has been incorporated, a structural stage could restrict and guide the type and number of inferences. Generally speaking, recognition of a structural stage enhances the attention focusing mechanism through a greater degree of system

intentionality. The reader is reminded that the current implementation of KBAM lacks the above sophistication.

Acquisition Processes

KBAM's theory of acquisition by explanation has been embodied within knowledge structures called Knowledge-Building Rules (KB-RULES). There is a one-to-one relationship between the different types of explanatory utterances presented and the KB-RULES. Each KB-RULE is mutually exclusive and represents one acquisition process. The KB-RULES are addressing two theoretical issues:

1. A small number of different types of explanatory utterances are sufficient in explaining a specific knowledge domain.

2. Each type of explanatory utterance is processed differently, resulting in a number of acquisition processes.

An actual KB-RULE is not one specific rule but several TASKS and RULES - CONTROL and PERFORMANCE UNITS. Each acquisition process is directed or guided by a CONTROL UNIT while the actual construction of new knowledge is performed by PERFORMANCE UNITS. Although each KB-RULE represents a different acquisition process, this does not mean that the same CONTROL or PERFORMANCE UNITS cannot be reused. Since each UNIT has been designed independent of the other, the UNITS can be applied amongst the KB-RULES whenever possible, however, the top-level CONTROL UNITS of each KB-RULE will always be unique. More often than not, it is the PERFORMANCE UNITS that are shared. Reflecting upon the functions of performance units, this would be expected since there are only a small number of data objects to modify (i.e., working memory or the knowledge-base).

A BRIEF EXPLANATORY PRESENTATION

As described previously, KBAM's principal resource is a conceptual analyzer. The example below is presented for the purpose of familiarizing the reader with the interaction between KBAM and the conceptual analyzer, as well as some of KBAM's acquisition processes.

The brief training sequence explains the name of a particular course (in this case, Computer-Science 130). It is but a subset of a much more extensive explanation presented to KBAM during this research. We are all familiar with the fact that the naming of concepts (the use of a lexical identifier) is a basic element in our communication process. Conversationalists often reference clusters of related knowledge (a concept, entity or episode) by a name. As a result, communication becomes more efficient. Therefore, it would seem necessary for an acquisition tool to have the ability to acquire both the conceptual content and the name (lexical identifier) of the concept.

In the example below, words which are capitalized within the explanatory utterances represent unknown concepts - those which must be acquired. Lower case words represent concepts whose semantics can be accessed by the analyzer before any acquisition occurs. They represent primitive concepts. For the purposes of this example, as concepts become acquired they will remain capitalized in all succeeding utterances. This should allow the reader to see at a glance which concepts are primitive and which concepts are acquired and used further along in the training sequence.

1. a COURSE is an episode.

2. a DEPARTMENT is an organization.

3. a DEPARTMENT and a number are part of the label of a COURSE.

4. COMPUTER-SCIENCE is the label of a DEPARTMENT.

5. 130 and COMPUTER-SCIENCE are part of the label of a COURSE.

1. a COURSE is an episode.

In this first utterance, the word COURSE has no semantic value to the conceptual analyzer or to KBAM. The user is attempting to introduce a new concept (COURSE) with respect to KBAM's present knowledge (episode is considered primitive knowledge). The utterance is presenting a generic concept (episode) and using it to define a specific concept (COURSE). The specific types of episodes defined from the generic episode are classified domain specific within this research. In this way, a domain specific concept is acquired and stored in the KB and the lexicon of the conceptual analyzer.

KBAM's primitive knowledge of episode includes a semantic representation and several associated assertions. These assertions embody knowledge which might be inferred otherwise. Some example assertions are: ANIMATE ENTITIES PARTICIPATE IN EPISODES and EPISODES ARE COMPRISED OF TEMPORAL RELATED EVENTS. The collection of associated assertions and any assertions acquired from the input utterance are stored in the KB. In this case, the associated assertions given above are instantiated with the specific concept name, COURSE, producing the assertions: ANIMATE ENTITIES PARTICIPATE IN COURSES and COURSES ARE COMPRISED OF TEMPORAL RELATED EVENTS. At the conclusion of this acquisition process the knowledge-base will not only have the original primitive knowledge concerning episodes but also a semantic representation for COURSE and a number of associated assertions pertaining to COURSE. This knowledge is needed prior to the presentation of further, more detailed explanatory utterances concerning courses; for example, those explanatory utterances needed in naming a course.

2. a DEPARTMENT is an organization.

Once again an explanatory utterance defining a domain specific concept (DEPARTMENT) with respect to a generic primitive concept (organization) is presented. The primitive semantic knowledge and assertions associated with organization, such as: ORGANIZATIONS CAN ACT AS AUTHORITIES, is applied to DEPARTMENT during the acquisition process. KBAM will also construct a Semantic Update Demon (SUD) for the organization concept. The SUD enables KBAM to perform future updates to any concepts which have been previously defined with respect to more general concepts. The update occurs whenever new information is acquired concerning the more general concept. That is, knowledge can be distributed from one explanation to the next. In the previous explanatory utterance in this example, a SUD for episode would also have been constructed. In the case where the SUD is created for a primitive concept, such as here, there is little chance of it being executed, however, the user may decide to introduce such information at a future time. If so, updating the relevant acquired concepts does not pose a problem.

3. a DEPARTMENT and a number are part of the label of a COURSE.

Explanatory utterances can now take advantage of previously acquired concepts. In this utterance, DEPARTMENT and COURSE are two such concepts. In this particular domain (i.e., the academic environment), the naming of courses is somewhat more difficult than naming an object such as a house or a ball. This is due to the fact that course names reflect both the department in which the course is offered and a unique code identifier. The present utterance is an attempt to describe this duality to KBAM. The acquisition process will utilize this input in a twofold manner. Initially KBAM will update its concept for COURSE (within the KB and the lexicon) by including two new characteristics COURSE-DEPARTMENT and COURSE-NUMBER. This is followed by the activation of a semantic restriction procedure which will enable KBAM to perform future semantic validity checks based upon these new characteristics. For example, in future explanations attempting to name courses, should a lexical item other than a number be presented, the validity checking knowledge would recognize the discrepancy.

4. COMPUTER-SCIENCE is the label of a DEPARTMENT.

In this utterance a generic department is being utilized in the formation of a specific department. This is accomplished simply by associating a name to the generic department. A new concept, COMPUTER-SCIENCE, is created which will inherit all the characteristics associated with DEPARTMENT. The reader should be careful not to assume that the COMPUTER-SCIENCE concept is inheriting only the characteristics of an ORGANIZATION (since DEPARTMENT had previously inherited those very characteristics). Although not part of this training sequence, more knowledge could have been acquired concerning DEPARTMENT after its conception.

Whatever KBAM's current concept of DEPARTMENT may be, this is the information that is inherited by the concept COMPUTER-SCIENCE. At the completion of this acquisition process a SUD for DEPARTMENT is constructed (assuming, of course, that one had not been previously constructed).

5. 130 and COMPUTER-SCIENCE are part of the label of a COURSE.

This utterance is the culmination of the user's efforts. It is important to note that the order of presenting 130 and COMPUTER-SCIENCE is inconsequential. The concept associated with COURSE is retrieved from memory in order to instantiate the characteristics COURSE-DEPARTMENT and COURSE-NUMBER. The acquisition process will first perform a validity check against the semantic restrictions for each characteristic (recall that these restrictions were created during a previous acquisition). This means that the lexical item 130 is being checked as a valid number and COMPUTER-SCIENCE as a valid department. If these lexical items prove to be valid "fillers" for the characteristics, the new concept for "COMPUTER-SCIENCE 130" is created, otherwise, the user is confronted with the error (the presentation of an invalid filler for the characteristic).

This example shows, in generalities, some of the primitive assertions, acquired assertions and the concepts formed by KBAM during acquisition. Besides placing these acquired pieces of knowledge into the knowledge-base, KBAM also stores each explanatory utterance (in its conceptual form) into the knowledge-base.

SUMMARY AND CONCLUSIONS

The most natural and practical manner for a person to communicate to a computer is through a natural language. Since explanations are an integral part of our daily lives, it follows that their use would enhance the human/computer interaction. With this in mind, we have designed and implemented a computer acquisition tool (KBAM) which accepts information in the form of explanations and develops a knowledge-base for a target expert system.

By investigating explanations and discovering their structural content, as well as the types of utterances comprising the explanation, it was possible to develop a theory of acquisition centered upon explanation as input.

Future developments of our mechanism will be directed toward the exploration and greater utilization of the structural stages encountered within explanations. As described, the stages can provide us with a greater understanding of intentionality and attention focusing. These new insights can then be designed into KBAM, enhancing its acquisition processes.

Further use and testing of KBAM with different domains will strengthen the current KB-RULES and further disclose the acquisition processes associated with each utterance type. Any need for

additional utterance types and, therefore, acquisition processes will also become obvious.

We can conclude that computer acquisition of a knowledge-base can be accomplished in a user-friendly manner. It is also possible to vary the domain of interest for different target expert systems providing that the conceptual analyzer's lexicon is supplied with the appropriate world knowledge primitives and the acquisition tool is supplied with associated world knowledge assertions. With time, the accumulation of this domain independent knowledge will reach a level of sufficiency for most future domain acquisitions. Should it be necessary to develop new acquisition processes as new domains are encountered, KBAM's production system architecture provides for easy modification and incremental development, and its attention focusing mechanism alleviates the burdensome production conflict problem.

REFERENCES

Anderson, J.P. (1981) A Theory of Language Acquisition Based on General Learning Principles, Proceedings of the International Joint Conference on Artificial Intelligence, p. 97.

Andrea, J. & Andrea, P. (1979) Machine Learning With a Multiple Context, Proceedings of the International Conference on Cybernetics and Society.

Birnbaum, L. & Selfridge, M. (1979) Problems in Conceptual Analysis of Natural Language, Research Report #168, Yale University.

Bransford, J.D. (1979) HUMAN COGNITION - LEARNING, UNDERSTANDING AND REMEMBERING, California : Wadsworth.

Bruner, J., Goodnow, J., & Austin, G. (1956) A STUDY OF THINKING, N.Y. : John Wiley & Sons.

Charniak, E. & Wilks, Y. (Eds.) (1976) COMPUTATIONAL SEMANTICS, N.Y. : North-Holland Publishing Co.

Charniak, E. (1982) A Common Representation for Problem-Solving and Language-Comprehension Information, Technical Report CS-59, Brown University.

Cohen, P.R. & Feigenbaum, E.A. (1982) THE HANDBOOK OF ARTIFICIAL INTELLIGENCE (vols. 1, 2, & 3), Los Altos, Cal. : William Kaufman, Inc.

Collins, A. & Stevens, A.L. (1982) A Cognitive Theory of Interactive Teaching, C.M. Reigeluth (Ed), INSTRUCTIONAL DESIGN THEORIES AND MODELS: AN OVERVIEW, N.J. : Lawrence Erlbaum.

Cullingford, R.E. & Pazzani, M.J. (1980) Word Meaning Selection in Multiprocess Language Understanding Programs, TR-CS-80-12A, University of Connecticut, Department of Computer Science.

Davis, R. (1978) Knowledge Acquisition in Rule-Based Systems-Knowledge About Representations as a Basis for System Construction and Maintenance, Waterman, D.A. & Hayes-Roth, F. (Eds), PATTERN-DIRECTED INFERENCE SYSTEMS, N.Y., San Francisco, & London : Academic Press, p. 99.

McDermott, J. & Forgy, C. (1978) Production System Conflict Resolution Strategies, Waterman, D.A. and Hayes-Roth, F. (Eds), PATTERN-DIRECTED INFERENCE SYSTEMS, N.Y., San Francisco, & London : Academic Press, p. 177.

Michalski, R.S., Jaime, C.G. & Mitchell, T.M. (1983) MACHINE LEARNING - AN ARTIFICIAL INTELLIGENCE APPROACH, Palo Alto, Cal. : Tioga Publishing Co.

Newell, A. (1973) Production Systems: Models of Control Structures, S. Chase (Ed), VISUAL INFORMATION PROCESSING, N.Y. & London : Academic Press, p. 480.

Riesbeck, C.K. (1975) Conceptual Analysis, R. Schank (Ed), CONCEPTUAL INFORMATION PROCESSING, Amsterdam/New York : North-Holland/American Elsevier, p. 83.

Rychener, M.D. & Newell, A. (1978) An Instructable Production System: Basic Design Issues, Waterman, D.A. & Hayes-Roth, F. (Eds), PATTERN-DIRECTED INFERENCE SYSTEMS, N.Y. & London : Academic Press, p. 135.

Schank, R.C. (1973) Identifications of Conceptualizations Underlying Natural Language, R. Schank & K. Colby (Eds), COMPUTER MODELS OF THOUGHT AND LANGUAGE, San Francisco : W.H. Freeman, p. 187.

Schank, R.C. & Abelson, R.P. (1977) SCRIPTS PLANS GOALS AND UNDERSTANDING, Hillsdale, N.J. : Lawrence Erlbaum Associates.

Underwood, G. (1978) Concepts in Information Processing Theory, G. Underwood (Ed), STRATEGIES OF INFORMATION PROCESSING, N.Y. & London : Academic Press.

Young, R.M. (1979) Production Systems for Modelling Human Cognition, D. Miche (Ed), EXPERT SYSTEMS IN THE MICRO ELECTRONIC AGE, Edinburgh University Press.

Winograd, T. (1972) UNDERSTANDING NATURAL LANGUAGE, N.Y. : Academic Press.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

Toward Automating Recognition of Differing Problem-Solving Demands

Jeffrey Stout, Gilbert Caplain,[1] Sandra Marcus,[2] and John McDermott[3]
Department of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

SALT provides a knowledge acquisition framework for the development of expert systems that use propose-and-revise as their problem-solving method. These systems incrementally construct a tentative plan, identify constraints on the plan and revise plan decisions in response to constraint violations. By having an understanding of the specific problem-solving method used to integrate the knowledge it acquires, it has been previously shown that SALT possesses a number of advantages over less restrictive programming languages. We have applied SALT to a new type of propose-and-revise task, and have identified several areas where SALT was too restrictive to adequately permit acquisition of domain knowledge. Focusing on these problems has led both to a more "general" SALT and to a better understanding of when it is an appropriate tool. It has also led to the recognition, within the propose-and-revise class of domains, of two sub-classes which make different demands on SALT, and toward a SALT which is able to distinguish between them.

1. Introduction

SALT[4] is a knowledge acquisition tool designed to assist in building expert systems that assume a propose-and-revise problem-solving strategy. A SALT-generated expert system will construct, as opposed to select, a solution (cf., [Clancey 85]). The SALT-assumed method incrementally builds up a description of a proposed plan, identifies constraints on parts of the plan and modifies

---

[1]Now at CERMA-ENPC, La Courtine B.P. 105, 93194 Noisy le Grand, France

[2]Now a principal researcher for the Advanced Technology Center, Boeing Computer Services, P.O.Box 24346, M/S 7L-64, Seattle, WA 98124.

[3]Currently on leave and working at Digital Equipment Corporation, HL02-3/CO7, 77 Reed Rd, Hudson, MA 01749

[4]kNowledge ACquisition Language

the proposal in response to the detection of constraint violations. As with other similarly motivated knowledge acquisition tools [Davis 82, Boose 84, Kahn 84, van de Brug 86, Eshelman 86], its ability to provide assistance derives from the fact that all the systems it builds use the same problem-solving method.

Giving a knowledge acquisition tool an understanding of the knowledge roles in the problem-solving method it presupposes gives it considerable leverage [McDermott 86, Clancey 85, Neches 85, Swartout 83]. This understanding results in a narrow, structured language which the domain expert can use to concisely convey knowledge to the tool and which the generated system uses to explain what it knows. It also helps in defining when that knowledge should be brought to bear in problem-solving, and in determining the adequacy of the knowledge base for its intended function. Finally, it determines which domains the tool can be used for. SALT fits all of these descriptions [Marcus 85, Marcus 86a], but it was not clear if it was too restrictive, unnecessarily sacrificing some generality.

SALT has been successful in acquiring the knowledge for and in generating VT, an expert elevator configurer [Marcus 86b], but it was not clear from that work to what degree both presupposing a single problem-solving method and basing that method on only a single domain restricted the scope of SALT's applicability. This paper details the "extension" of SALT to a new domain, that of production scheduling. This extension has led to a SALT which can handle a wider range of constructive tasks (without compromising any of its leverage), and has also helped to better define when SALT and a knowledge domain are appropriate for each other. More importantly, however, we have found that no single propose-and-revise scheme is appropriate for all such tasks. Thus, SALT must be able to determine which scheme is best for a particular domain and act accordingly.

The next section briefly reviews the method SALT assumes. Section 3 describes the scheduler which was built using SALT. Section 4 discusses how SALT had to be changed, and the implications of those changes both for the problem-solving method and the scope of tasks for which SALT is appropriate.

2. The Method SALT Assumes[5]

A SALT-generated system constructs, rather than selects (cf., [Clancey 84, Clancey 85]), solutions using a "propose-and-revise" method. In many domains, it is not possible to construct a solution by a simple sequence of deterministic decisions. In such domains, plausible reasoning combined with the ability to backtrack to revise bad guesses can be an efficient problem-solving strategy [Stefik 83]. Even least commitment planners such as MOLGEN [Stefik 81a, Stefik 81b] sometimes have a need for this kind of reasoning. For example, decisions in the course of constructing a solution are sometimes underconstrained, or must be made based on incomplete information. The method SALT assumes allows for a least commitment construction of a proposed plan, but offers the capability of revising decisions when the proposed plan does not

[5]See [Marcus 86a] for a more complete discussion of SALT.

meet all constraints.

SALT categorizes domain knowledge according to the role it plays in problem-solving. From SALT's perspective, domain-specific knowledge always plays one of three roles:

1. PROPOSE-A-PLAN-EXTENSION
2. IDENTIFY-A-CONSTRAINT on a plan extension
3. PROPOSE-A-FIX for a constraint violation

Each "piece" of PROPOSE-A-PLAN-EXTENSION knowledge specifies a procedure for defining the value of a part of the plan description. These procedures may either determine that value with certainty or propose a preferred choice. (The user need not be aware of this distinction.) Details of the procedures consist largely of constants to be assigned, algebraic formulas for calculations and specifications which are translated into database calls. These procedures are also used to specify IDENTIFY-A-CONSTRAINT knowledge. Additional knowledge is required in declaring the value to be a constraint, identifying the value it constrains, and in specifying how it limits that value (e.g., a minimum or maximum). PROPOSE-A-FIX knowledge identifies a decision that could be changed in order to remedy a particular constraint violation, the nature of that change and some indication of how costly the change would be.

Classifying knowledge in this way categorizes it according to the effect it has in determining the values of variables that represent the final plan description (the desired output of the system). PROPOSE-A-PLAN-EXTENSION proposes a value for a variable, IDENTIFY-A-CONSTRAINT limits the value of a variable and PROPOSE-A-FIX revises the value of a variable. Organizing the knowledge base in this way still leaves open the question of how these pieces of knowledge fit together to form a sequence of actions that constructs a solution. An outline of how they fit into the method is given below:[6]

1. Propose extensions to the plan and identify constraints until a constraint violation is detected.
2. Propose possible remedies for the constraint violation.
3. Select the least costly fix not yet attempted.
4. Tentatively revise the plan insofar as to examine the effect on the constraint violation.
5. If the constraint violation persists, go back to Step 3.
6. Remove portions of the plan inconsistent with the revision.

---

[6]The SALT-generated problem-solving architecture is quite similar to that of EL [Sussman 77, Stallman 77], although it differs somewhat in how the system decides which decisions to revise.

7. Implement the revision and return to Step 1.

Proposed plans are built incrementally in the sense that each step to extend the plan uses input or decisions already made. Simultaneously, the system identifies constraints on the plan as enough information becomes available to derive their values. The control in this phase is data-driven; whenever all the information required to apply a procedure is available, the procedure is applied. (More than one procedure may be "eligible" at a time, but their relative order is unimportant.) As each procedure is applied, a dependency network is built up that records the specifics of the applied procedure. Constrained values are compared with their constraints as soon as both are proposed. If a constraint violation is detected, the system will use domain-specific proposals (fixes) to determine which decisions may be revised. It also uses domain-specific knowledge about the relative cost of revisions plus a measure of likelihood of success in deciding which revision(s) to try. Once a successful fix is found, a truth maintenance system is used to trace through the dependency network and remove any values that may be inconsistent with the change(s) made. The system then re-enters the data-driven phase of extending the proposed plan using the new data.

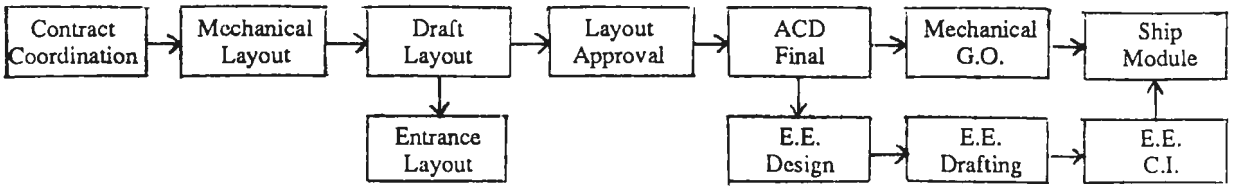
3. Building the Scheduler

3.1. Selecting the Factory Scheduling Domain

SALT first crystalized during the building of VT, an expert system for configuring elevators [Marcus 86b]. VT takes as input the customer's functional specifications for the elevator, such as the speed at which it must travel, its carrying capacity, and key architectural information about the elevator shaft, such as the wall-to-wall dimensions and locations of sills and rail supports. The task is to select all the necessary pieces of equipment and produce a configuration and layout of parts that meet those specifications, as well as safety, installation, and maintenance requirements.

Although SALT was successful in building and maintaining VT, a single case gives little indiction of a tool's generality or the range of tasks over which it is applicable. Thus the domain of factory scheduling was chosen as a task clearly different from electromechanical system configuration that SALT should be able to do. It appeared to be viable since all that SALT really seemed to require of a task was that it could be structured to provide methods for initial values for extending the plan, methods for defining boundaries between acceptable and unacceptable solutions, and a way to change values in response to a particular boundary violation that will lead to an acceptable solution.

Given this view of SALT's assumed method, scheduling appeared to be a suitable test-domain, since many scheduling tasks require coming up with an initial schedule, recognizing situations that make the schedule infeasible, and making minimally disruptive changes which make the schedule acceptable once more. To test this suitability, we built a prototype for production scheduling for escalators and elevators. The table below shows the flow of work performed to produce and deliver an order for one product line:



Each box represents a department that must do some processing on an order of a given product type. Each department must complete work on an order before it can move on to the next department (indicated by an arrow). Departments which are able to process an order in parallel lie on different branches. Associated with each department is a normal "lead" time for a given product type. This represents the amount of time the order would normally remain in a department while work was being done on that job. Not all of the lead time is actual hands-on processing time. For example, in the Mechanical Layout department the engineer might normally review a job as soon as it is assigned, checking for any missing information required from the customer or from other departments simultaneously processing the job, putting out routine requests for information, and then working on other orders while those requests are filled. There is a possibility for "shrink" time, requiring a job to be completed in the minimal amount of hands-on time. "Slack" time, leaving a job in a department longer than necessary, is also used to do things such as smoothing out the load to subsequent departments or avoiding shipping equipment to the field before the field is prepared to install it. The task of the scheduler, then, is to provide start and complete dates for each relevant event in the processing of each order.

3.2. Acquiring the Knowledge

When giving SALT a piece of domain knowledge, the user must first indicate what role that knowledge will play. Once a role is selected, SALT presents the user with a schema containing slots which the user may fill to complete the specification of that piece of knowledge. The three examples below are representative pieces of scheduler knowledge which fill each of SALT's three knowledge roles.[7] More specifically, they show completed SALT schemas.

The first example is a piece of PROPOSE-A-PLAN-EXTENSION knowledge.

```
1  Name:                          START-DATE
2  Defined for:                   DEPT ORDER
3  Precondition:                  [ DEPT = SHIP-MODULE ]
                                     AND [ PRODUCT-CODE = JB-2 ]
4  Procedure:                     CALCULATION
5  Formula:                       MAX [ COMPLETE-DATE@DEPT=MECH-GO
                                     COMPLETE-DATE@DEPT=EE-CI ]

Enter your command [EXIT]:
```

It should be noted that DEPT and ORDER are defined as indices, which means that there may be multiple values for START-DATE, with each one specific to a particular department name and order number. COMPLETE-DATE@DEPT=MECH-GO means the COMPLETE-DATE whose

[7]For a more thorough discussion, see [Marcus 87].

department index is MECH-GO.[8]

SALT translates the above information into a PROPOSE-A-PLAN-EXTENSION rule which can be paraphrased as follows:

> Given that the scheduled COMPLETE-DATE for an ORDER has been generated for the MECH-GO DEPT, and
>
> Given that the scheduled COMPLETE-DATE for that ORDER has been generated for the EE-CI DEPT,
>
> If there is no value for the STARTING-DATE for that ORDER for the SHIP-MODULE DEPT, and
>
> If the ORDER's JOB-TYPE is B2, then:
>
> Make the SHIP-MODULE STARTING-DATE for the ORDER the larger of the two COMPLETE-DATES.
>
> Leave a trace that the ORDER's JOB-TYPE and those COMPLETE-DATES contributed to this STARTING-DATE.[9]
>
> Leave a trace describing the inference as given here.[10]

Thus this procedure captures the domain knowledge, indicated by the flowchart, that processing in the SHIP-MODULE department begins when both the MECHANICAL-GO and EE-CI departments have finished with the order.

The next example is a piece of IDENTIFY-A-CONSTRAINT knowledge.

```
1  Constrained value:          COMPLETE-DATE
2  Constraint type:            MAXIMUM
3  Constraint name:            MAXIMUM-COMPLETE-DATE
4  Defined for:                DEPT ORDER
5  Precondition:               DEPT = SHIP-MODULE
6  Procedure:                  SPECIFIED-COMPLETION-DATE

Enter your command [EXIT]:
```

SALT translates this information into an IDENTIFY-A-CONSTRAINT rule which specifies a procedure for specifying a value for the constraint and provides crucial information about the constraint to the problem-solver. A paraphrase of this rule would be:

> Given that there is a SPECIFIED-COMPLETION-DATE for an ORDER (it is defined as an input value),
>
> If that ORDER has no value for MAXIMUM-COMPLETE-DATE for the SHIP-

---

[8]Indices are actually doing much more that this. COMPLETE-DATE, as one might suspect, is indexed by ORDER as well. SALT will assume (unless told otherwise) both that all ORDER indices for any values in the procedure indexed by ORDER must match, and that they determine the value of the ORDER index of the result.

[9]Leaving a trace of contribution builds the dependency network which is used by the truth maintenance system.

[10]This is used by the explanation facility.

MODULE DEPT, then:

Make the MAXIMUM-COMPLETE-DATE for the SHIP-MODULE DEPT for that ORDER the value of the SPECIFIED-COMPLETION-DATE.

Identify this value as a MAXIMUM constraint on the COMPLETE-DATE for the SHIP-MODULE DEPT for that ORDER.

Leave a trace that the SPECIFIED-COMPLETION-DATE contributed to this MAXIMUM-COMPLETE-DATE.

Leave a trace describing the inference as given here.

Since the user has identified a constraint, SALT will expect proposals for remedying violations of that constraint. The final example is just such a piece of PROPOSE-A-FIX knowledge.

```
1  Violated constraint:                MAXIMUM-COMPLETE-DATE
2  Value to change:                    LEAD-TIME@DEPT=*
3  Preference:                         MINIMIZE START-TIME
4  Change type:                        DECREASE
5  Step type:                          TO-CUTOFF
6  Rating of undesirability:
7  Reason for undesirability:

Enter your command [EXIT]:
```

This fix, and others for the same constraint violation, are translated into a PROPOSE-A-FIX rule which will suggest potential remedies whenever the associated constraint is violated. The paraphrase of this rule would be:

If there has been a violation of MAXIMUM-COMPLETE-DATE for an ORDER, then:

Try to DECREASE that ORDER's LEAD-TIME in any DEPT down to its minimum value (the "shrink" time). In selecting a DEPT, give preference to those with the smallest START-TIME.

4. SALT as an Appropriate Tool

Although SALT is now an appropriate tool for the scheduling task, a significant amount of effort was required to make it so. Slight changes in the representation were necessary, as well as variations in how domain knowledge is brought to bear. This section details those changes, and how those changes have helped in understanding both when SALT is appropriate and how it can determine for itself how it can be of the most help.

4.1. Changes Necessitated by the Scheduler

The development of SALT was originally driven by VT's needs; SALT was as general as VT demanded it to be. It could have been made more general, but it was apparent that there were an unlimited number of possible "generalizations", and it was unclear which were appropriate for the domains SALT would encounter. Thus SALT was kept very simple, perhaps even overly so. The scheduling task, although a constructive task, did not initially fit well within SALT's propose-and-revise scheme. Some of the reasons for this were:

- The regularity of domain values: Virtually all values in the system were closely related to others in a regular fashion. For example, there was a START-TIME, LEAD-TIME, and END-TIME for each order for each department it went through. Since the END-TIME was always the sum of the START-TIME and the LEAD-TIME, it would have been unrealistic to expect a SALT user to enter a method for each combination of these.

- The dynamic nature of fix preferences. VT's fix knowledge was static; the precise value to be fixed was specified in the SALT interview, and the relative cost of fixes were based on a static cost function. The values the scheduler needs to fix are uncertain. Such values had to be grouped statically, and a function was necessary to select one at run time.

- The highly interactive nature of fixes. For the scheduler, almost any fix has the potential of causing new constraint violations. VT resolves constraints as they arise, but the scheduler must be more careful. The VT-based strategy of addressing constraint violations as they occur would cause the system to deal with minor problems before recognizing that a larger problem exists that will make the other fixes moot. The scheduler must delay constraint checking until a full tentative schedule has been built, decide which violated constraint should be addressed first, and select the "least disruptive" fix.

- The lack of a need for lookahead. When the scheduler has selected a fix, it does not need to test it before implementing it. This is because a local search is not reliable in predicting success, the fix will never make things worse, and also because, if it did not succeed, the next choice would probably be to use it in conjunction with the next-least-costly fix.

The first two problems were straightforward generalizations of SALT. Indexing was a feature that had long been considered a reasonable extension. As VT continued to develop, it became obvious that such a feature would eventually be necessary if it was to reach the level of full-scale production. VT handled only single-car elevator banks; the extension to multi-car banks was not feasible without some type of indexing. Originally, the cost of a fix was selected from an ordered menu of negative effects. This has been generalized to allow the user to input a cost which may be a value specified by a procedure. In the case where there are multiple candidates, they can now be ordered based on an index. (This ordering is temporal for the scheduler, since it is usually best to fix things "as soon as possible".)

The scheduler's requirement that an entire plan be built before addressing constraint violations, in addition to it not needing lookahead, was in direct conflict with VT's needs. The difference between the two strategies was not a matter of knowledge; it was a difference in how knowledge is brought to bear by the problem-solving method. Thus the solution was to build two separate control shells which make different use of the same SALT-generated rules.

4.2. Determining when SALT is Appropriate

The key to determining the usefulness of a tool like SALT for a particular task is to ask whether its assumed problem-solving strategy is appropriate. It is therefore important to characterize SALT's assumed method with respect to the types of demands it makes on the knowledge sources of the domain. An initial "broad cut" can be made by the fact that the method constructs, rather than selects, a solution (cf., [Clancey 84, Clancey 85]). A selection method demands that the domain expert be able to pre-enumerate the entire set of possible solutions. For tasks which cannot meet this criterion, construction is the alternative.

It is in these cases where the domain fails the test for supplying a sequence of pre-enumerated decisions that propose-and-revise becomes a candidate. Propose-and-revise is certainly not the only candidate for addressing this failure (see [van de Brug 86]), but enough is not yet known about the others at this point to determine when it would be the best candidate. The propose-and-revise strategy makes important demands on the organization of the domain. More specifically, in order for SALT to be an appropriate tool, the domain expert must be able to describe the task by categorizing each piece of the domain knowledge into one of the three roles described previously.

4.2.1. SALT and VT

Because VT deals with a large number of potential part combinations and must customize the layout to the space available in individual buildings, it must construct its solution. VT cannot provide a completely predetermined sequence of decisions; many of the decisions it must make are underconstrained. For example, the width of the elevator car plus the width of the counterweight plus the space between the car and counterweight plus the space between the counterweight and the wall must sum to the wall-to-wall distance in the elevator shaft. The width of the car is selected by the customer, but this input combined with the constraint do not result in an algorithm for determining the other three values. VT also has cases of decisions so interdependent that no decision can be made without knowing its own result. For example, the hoist cable quantity and diameter depend on hoist cable quantity and diameter -- they must be able to support their own weight, as well as the weight of other parts whose selection is based partly on the hoist cable quantity and diameter.

VT's domain experts are able to cast their knowledge into the roles that SALT assumes. VT is able to use propose-and-revise for the cases described above. For example, the VT method estimates the lowest possible value for hoist cable quantity and diameter (PROPOSE-A-PLAN-EXTENSION), selects other parts using these estimates (also using PROPOSE-A-PLAN-EXTENSION knowledge). It then derives from these a constraint on the quantity and diameter that *must* be used (IDENTIFY-A-CONSTRAINT). If the value of the constraint does not match the initial estimate, quantity and diameter are increased (PROPOSE-A-FIX).

4.2.2. SALT and the Scheduler

The scheduler must construct its solution because at any one time there will be a unique configuration of both orders in the system and available resources that will affect the assignment of dates. It is impossible for the scheduler to operate based on a predetermined sequence of decisions, since many of its decisions are underconstrained, and because of changes in the world

as new orders are entered and resources shift (due to shortages, sick days, and so forth).

Propose-and-revise is a reasonable candidate to address this problem, and the SALT knowledge roles are appropriate as well. The flow chart shown previously is a high-level view of the procedures used to propose a tentative schedule for an individual order using normal lead times (PROPOSE-A-PLAN-EXTENSION). There are several ways that an initial schedule may be unacceptable (IDENTIFY-A-CONSTRAINT). For example, contracts may have promised dates for specific events, and departments have a maximum work load. The responses to these situations include rescheduling orders and shifting resources (PROPOSE-A-FIX).

5. Conclusion

The scheduling task was initially chosen both because it was obviously a configuration task, and because it was not so obviously a propose-and-revise task. By allowing SALT to change in response to the needs of the task, we have developed a more general SALT. If any of its leverage as a method-based knowledge acquisition tool was sacrificed in the process, it was leverage of a type yet to be discovered, since VT can not only still be built and maintained using SALT, it can now be done more easily and can be extended to cover a larger portion of elevator jobs.

In determining whether SALT and a domain are appropriate for each other, one should not become bogged down in the specifics of the problem-solving strategy. SALT is appropriate for the scheduling task, not because the task had the exact same form as that of VT, but because the knowledge roles were well defined and matched those which SALT assumed. If a prospective domain has PROPOSE-A-PLAN-EXTENSION, IDENTIFY-A-CONSTRAINT, and PROPOSE-A-FIX knowledge classes (and only those), it should be possible to either use an existing control shell (from another domain) or modify one of the existing control shells to handle the unusual features of the new domain.

It is possible that a domain could have the three SALT knowledge roles but *not* fit into a propose-and-revise scheme. Such domains would be invaluable in bounding the set of problems appropriate for SALT. However, one has yet to be discovered.

As more variations on propose-and-revise are identified, it should become more obvious how SALT can automatically determine, on the basis of the knowledge base, which control shell to use. Currently, SALT handles this by providing essentially a user-operated switch for selecting a control shell. Ideally, SALT should understand and detect the conditions under which variations in the propose-and-revise strategy are required so that it can operate the switch itself. Such an ability will require an understanding of how, in a given domain, constraint violations interact to converge on a solution. The first step is to have SALT perform an analysis of the constraint/fix network in order to understand the nature and degree of the interrelatedness. We have experimented with VT's domain to develop such analysis techniques [Marcus 86b] and have plans for automating this analysis. As more domains are "acquired" by SALT, a clearer picture should emerge of what it means for a task to be called "propose-and-revise".

References

[Boose 84] Boose, J.
Personal construct theory and the transfer of human expertise.
In *Proceedings of the Fourth National Conference on Artificial Intelligence.* Austin, 1984.

[Clancey 84] Clancey, W.
Classification problem solving.
In *Proceedings of the Fourth National Conference on Artificial Intelligence.* Austin, 1984.

[Clancey 85] Clancey, W.
Heuristic classification.
Artificial Intelligence (27):289-350, 1985.

[Davis 82] Davis, R. and D. Lenat.
Knowledge-Based Systems in Artificial Intelligence.
McGraw-Hill, 1982.

[Eshelman 86] Eshelman, L. and J. McDermott.
MOLE: A Knowledge Acquisition Tool That Uses Its Head.
In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 950-955. Philadelphia, 1986.

[Kahn 84] Kahn, G., S. Nowlan, and J. McDermott.
A foundation for knowledge acquisition.
In *Proceedings of IEEE Workshop on Principles of Knowledge-based Systems.* Denver, Colorado, 1984.

[Marcus 85] Marcus, S., J. McDermott, and T. Wang.
Knowledge acquisition for constructive systems.
In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 637-639. Los Angeles, 1985.

[Marcus 86a] Marcus, S., and J. McDermott.
SALT: A Knowledge Acquisition Tool for Propose-and-Revise Systems.
Technical Report CMU-CS-86-170, Carnegie-Mellon University, Department of Computer Science, 1986.

[Marcus 86b] Marcus, S., J. Stout, and J. McDermott.
VT: An Expert Elevator Configurer that Uses Knowledge-Based Backtracking.
Technical Report CMU-CS-86-169, Carnegie-Mellon University, Department of Computer Science, 1986.

[Marcus 87] Marcus, S., G. Caplain, J. McDermott and J. Stout.
Making SALT generic.
Technical Report, Carnegie-Mellon University, Department of Computer Science, to appear, 1987.

[McDermott 86] McDermott, J.
Making expert systems explicit.
In *Proceedings of the IFIP Congress.* Dublin, Ireland, 1986.

[Neches 85] Neches, R., W. Swartout, and J. Moore.
Explainable (and maintainable) expert systems.
In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 382-389. Los Angeles, 1985.

[Stallman 77] Stallman, R.M., and G.J. Sussman.
Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis.
Artificial Intelligence (9):135-196, 1977.

[Stefik 81a] Stefik, M.
Planning with constraints (MOLGEN: Part 1).
Artificial Intelligence (16):111-140, 1981.

[Stefik 81b] Stefik, M.
Planning and meta-planning (MOLGEN: Part 2).
Artificial Intelligence (16):141-170, 1981.

[Stefik 83] Stefik, M., J. Aikins, R. Balzer, J. Benoit, L. Birnbaum, F. Hayes-Roth, and E. Sacerdoti.
The architecture of expert systems.
In Hayes-Roth, F., D. Waterman and D. Lenat (editor), *Building Expert Systems*, pages 89-126. Addison-Wesley Publishing Company, Inc., 1983.

[Sussman 77] Sussman, G.J.
Electrical design, a problem for artificial intelligence research.
In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 894-900. Cambridge, 1977.

[Swartout 83] Swartout, W.
XPLAIN: A system for creating and explaining expert consulting systems.
Artificial Intelligence (21):285-325, 1983.

[van de Brug 86] van de Brug, A., J. Bachant and J. McDermott.
The Taming of R1.
IEEE Expert Vol. 1, No. 3, 1986.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

A KNOWLEDGE ACQUISITION ENVIRONMENT FOR SCENE ANALYSIS

Deborah Tranowski
Advanced Decision Systems
201 San Antonio Circle
Mountain View, CA 94040
debbie @ ads.arpa

ABSTRACT

This paper describes a knowledge acquisition environment under development to help capture expertise from domain experts involved in analyzing scenes from aerial imagery. The research is important because automated image understanding systems are increasingly relying on expert knowledge to help analyze objects and control the analysis process. It is desirable to enable the domain experts to enter and manipulate the domain knowledge directly. The research described is based on the concept of an integrated knowledge acquisition environment (KAE). The goal is to integrate the domain inputs, the translation into internal representations and the actual execution and feedback. The KAE contains a collection of computer-based tools facilitating: viewing and editing domain knowledge in both textual and graphic format (analysts tend to be visually oriented), knowledge base execution and testing, and expert system performance analysis.

1. INTRODUCTION AND RESEARCH MOTIVATION

A perceptual and reasoning activity of increasing importance is the capability to interpret and analyze imagery to detect objects, or activities of interest, to notice changes in a scene, or to interpret observed behaviors. This need is strongly identified in military applications of intelligence gathering such as predicting apparent presence and behavior, for verifying expected behavior (say for treaty verification). It is also important for a variety of other applications such as medical imagery diagnosis, mineral or crop analysis from aerial photographs, and industrial inspection.

In recent years, advances in sensor technology for imaging (for not only optical sensors but also radar, infrared, x-ray, laser etc.) have lead to a vast explosion in the quantity and quality of available imagery for the tasks mentioned above. As the imagery has become available, human "experts" have emerged that have remarkable perceptual abilities to locate and analyze objects in the image. However, they can not keep up; the imagery explosion has outrun manpower and training availability. This has lead to emphasis in developing automatic photo-interpretation systems.

Research in knowledge-based image understanding and vision systems to develop automatic photo-interpretation capabilities has focused on capturing human expertise and "rules" about how to identify objects or how to undertake and control the image analysis process. Hence there is a strong motivation to provide knowledge acquisition and refinement of knowledge for the vision system.

The nature of the photo-interpretating problem and the requirements of the knowledge acquisition process that drive an integrated knowledge acquisition capability include:

- The knowledge is visually oriented. Typically, visual appearance is being described. It may be the
  - o appearance of objects in certain conditions (in terms of their size, shape, texture, color, etc.)
  - o simple relationships between objects (such as near, adjacent, behind, on-top-of, etc.)
  - o more complex relationships between objects that describe a spatial pattern
- A tight coupling is required between the expert and the expert system (vision system) being developed. Because visual concepts are difficult to express, being able to suggest one and obtain rapid feedback on how the concept affected the system's performance is critical.
- Multiple types of expertise are often required and multiple experts are likely to be involved. For example, photo-interpreters, terrain experts, intelligence experts, and sensor experts may all participate in finding military units deployed in aerial radar imagery. The vision system consists of several cooperating expert systems each supporting knowledge bases and relevant knowledge representation methodologies. Hence methods are required for allowing each expert to contribute to, interact with, and obtain feedback from a larger, more complex, but unified system.
- The domain experts are likely to be fairly unsophisticated in working with computers. Often, asking the experts to discuss or describe computer representations of knowledge causes them to alter their knowledge and provide biased information. Hence, it is especially critical that all knowledge be described and discussed in domain terms and that computer science implementation issues be transparent.
- Currently, domain expertise is elicited through conversations (interviews) with experts. A knowledge engineer familiar with the system organization and representation schemes as well as possessing a degree of familiarity in the domain performs the role of intermediary between the computer system and domain expert. As the various domain knowledge bases are enhanced, the manual process of obtaining information from human experts and representing that information in a computer usable form is becoming increasingly cumbersome. It is desirable to reduce the dependence on the knowledge engineer and allow the domain experts to enter and manipulate the domain knowledge directly. This would also eliminate any biases that may be imposed by the knowledge engineer thus allowing the expert to describe their own cognitive techniques.

2. APPLICATION OVERVIEW

Driven by the motivations described above, a prototype Knowledge Acquisition Environment (KAE) has been developed. In order to make rapid progress in the prototype development and to provide near-term assistance with real photo-interpretation system development activities, these specific tasks involving scene analysis were selected for study. These tasks are:

- using terrain information and tactical analysis to determine likely locations for objects of interest; terrain analysis
- identification of objects based on their spatial layout; spatial pattern identification

Terrain analysis combines two types of expertise:

- Tactics concerning likely object locations and
- Knowledge of the underlying terrain in an area.

The target system (for which knowledge is being acquired) operates by defining constraints about locations and combining the constraints (in a weighted fashion) to obtain most likely overall locations. For example, constraints for predicting most likely locations for a regiment headquarters may include:

- behind hills (high ground)
- near (but not on) a road
- in a lightly wooded region
- fairly dry area

Spatial pattern identification involves analyzing a set of image derived objects to see if they are deployed in a pattern that is significant (reflects the presence of an activity or scene of interest). This analysis is possible because of tactical rules about how military units deploy and behave. Hence, the experts are military analysts. The target system attempts to locate an instance of a unit in a spatially distributed set of component units. The domain knowledge in the system specifies the constraints on the pattern of interest. There may be missing component units, additional false component units or incorrectly deployed (based on doctrine) component units in the scene.

3. KNOWLEDGE ACQUISITION ENVIRONMENT (KAE)

Given the problem scope described in the previous section, an initial step in the development of the prototype KAE was to define a flexible, yet robust architecture. The KAE architecture should provide support of a collection of computer-based tools facilitating:

- viewing and editing domain knowledge in both textual and graphic format (analysts tend to be visually oriented in their reasoning and interpretations)
- translation of raw domain information onto an intermediate representation and finally translation into an executable format
- knowledge base execution and testing
- expert system performance analysis
- knowledge base management

Figure 3-1 shows the KAE architecture. The user (domain expert or knowledge engineer) interacts with the KAE through the interface manager. Section 3.1 describes the tools and interactions currently available through the interface manager. Once entered into the KAE, raw domain information is translated into an intermediate representation (i.e., frames, rules). Section 3.2 explains the motivation for an intermediate representation and the types of representations available within the KAE. The domain information and data are then submitted to the appropriate inference engine for processing. Results are the output of the inferencing process. Results can then be analyzed and displayed to the user. Section 3.3 describes execution and analysis in the KAE. Having completed the cycle of input, translation, execution and display, the user has the opportunity to receive immediate feedback in terms of displays and analysis to guide the refinement process.

As the KAE begins to store and manage large bodies of domain knowledge, knowledge management tools will be essential. Section 3.4 presents an envisioned set of knowledge management tools.

Components of the interface manager and translation modules have been implemented for the terrain analysis and spatial pattern identification domains. The user interface allows information to be entered/edited in both a textual and graphical fashion as well as the ability to display information on maps and images for additional support.

Current development is being performed on a Symbolics 3270 with plans to migrate to a SUN workstation during 1988.

3.1 INTERFACE MANAGER

The goal of the KAE user interface is to support experts in the various domains required to perform scene analysis. In most instances, the domain experts are not sophisticated computer users, therefore an additional requirement is to make the interactions as easy and intuitive as possible. In order to meet these requirements

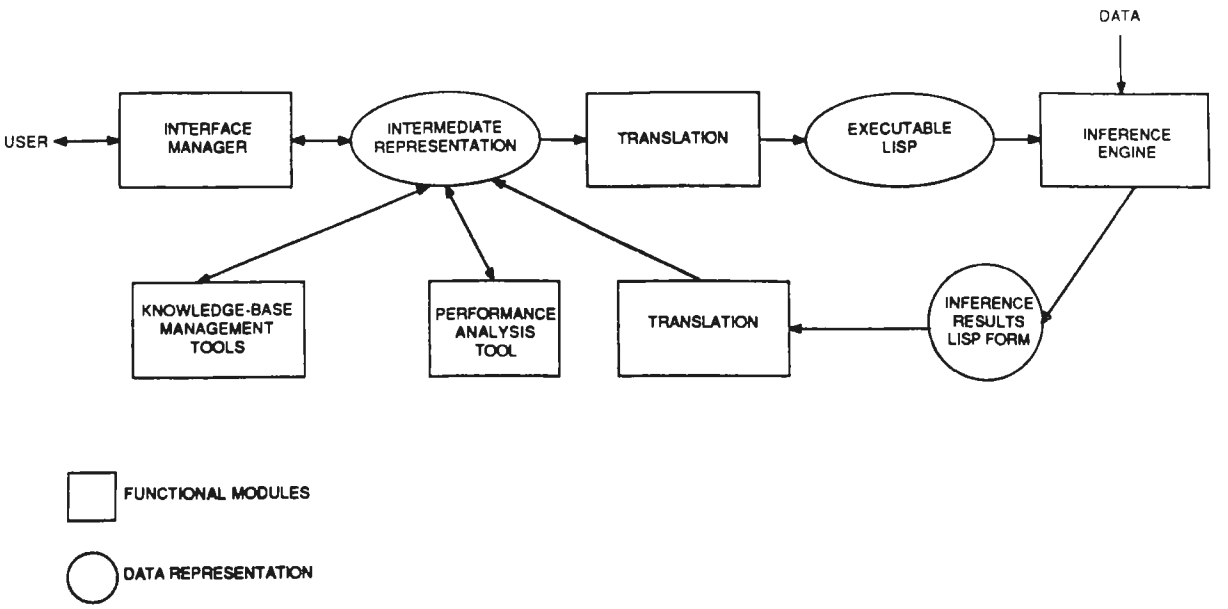


Figure 3-1: Knowledge Acquisition Environment Architecture

domain models [Musen - 86] were defined for the initial key areas (terrain analysis and spatial pattern identification). Domain models provide a means in which the intended actions, goals, and vocabulary of the application (i.e., terrain analysis) are formally expressed. By combining graphics and text within the specific application domain model the user is insulated from the detailed implementation and can interact with the information in ways which are familiar.

For example, during the process of defining the terrain analysis domain model it became apparent that experts use a variety of "overlays" to make assessments. An overlay represents a particular terrain feature (i.e., canopy closure) which is displayed over a map or image under analysis. A unit is then given a likelihood probability constituting suitability for each terrain feature. This process has been modeled in the KAE.

The KAE screen layout is presented in Figure 3-2. The top right window is the System Menu window. This window reflects the comprehensive set of information required to perform scene analysis. As previously stated, the prototype KAE is concentrating in the terrain analysis and pattern identification domain. The item selected from the System Menu provides further context-specific options that are then displayed in the sub-menu window. Selection of an item in the sub-menu performs a system processing operation. The displays generated by system processing are positioned in the Work Area. During the processing of a function, messages may appear in the Notification Window. As shown in Figure 3-3, selecting Terrain Analysis from the System Menu produces a sub-menu of actions specific to terrain analysis.

Once the user chooses to "Enter or Edit Unit Information" a knowledge base of previously defined unit specifications is read into the KAE. These are shown is Figure 3-3. The user decides to add a "new" unit to the knowledge base and is prompted for the unit name (Figure 3-4). The user is then prompted to select the

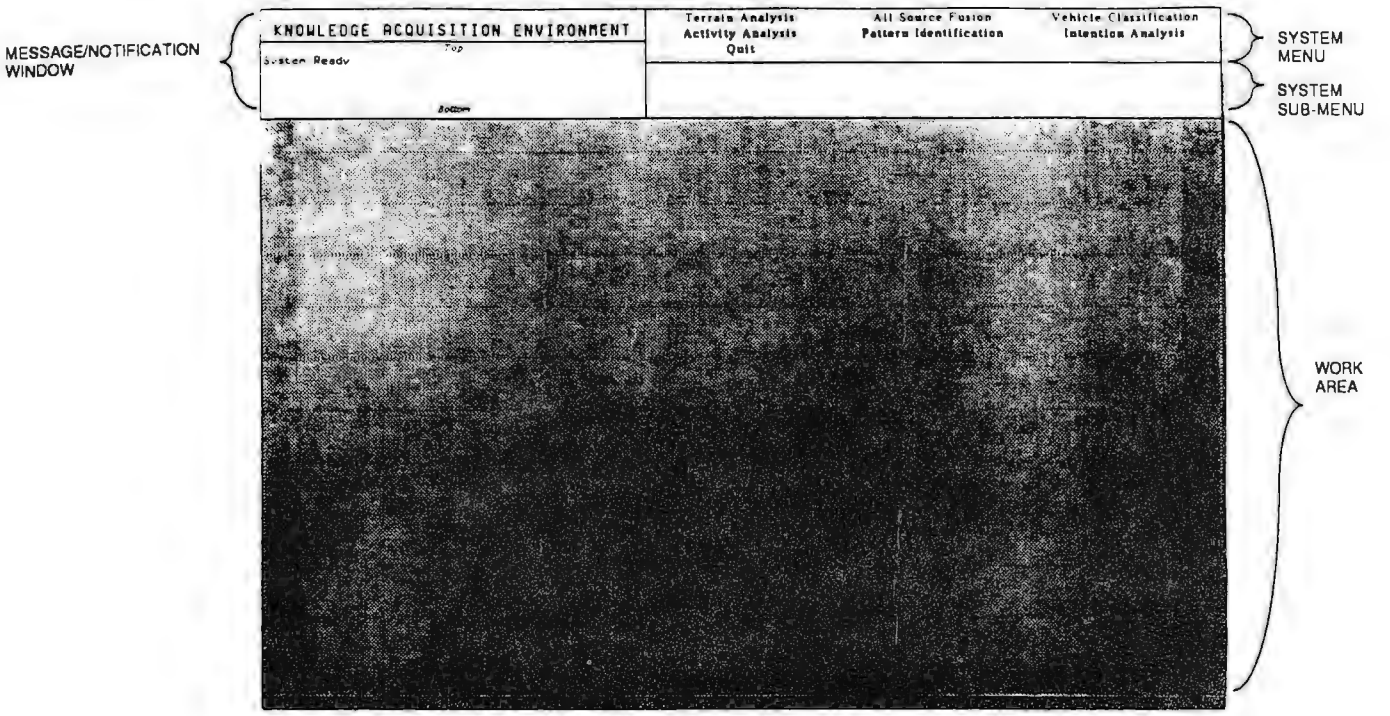


Figure 3-2: Knowledge Acquisition Environment - Screen Layout

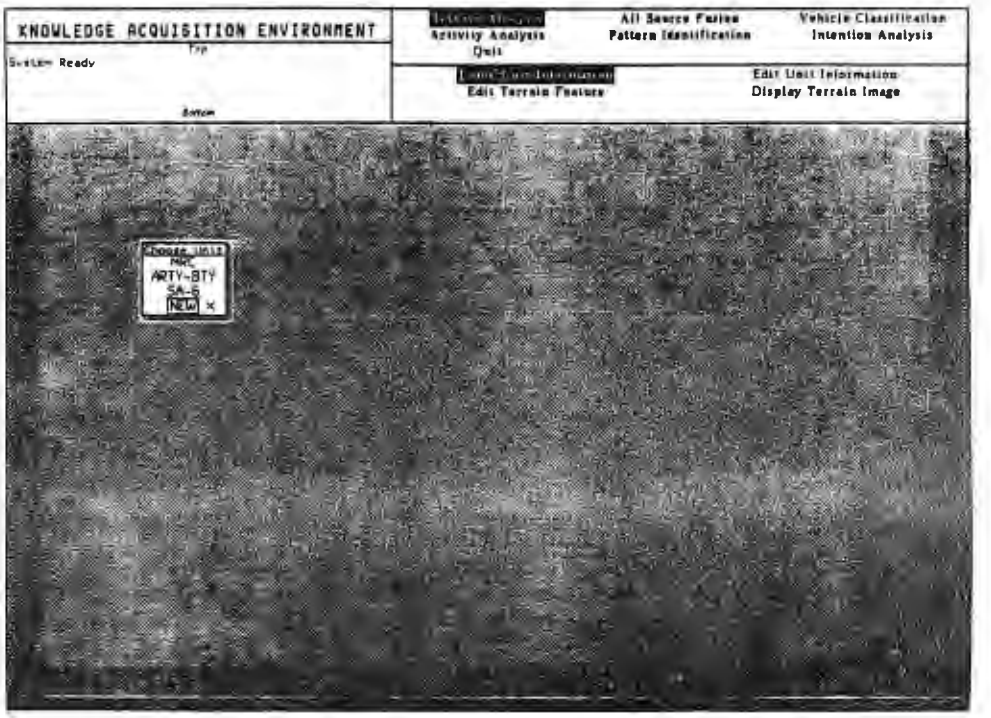


Figure 3-3: Adding a New Unit to the Terrain Analysis Knowledge Base

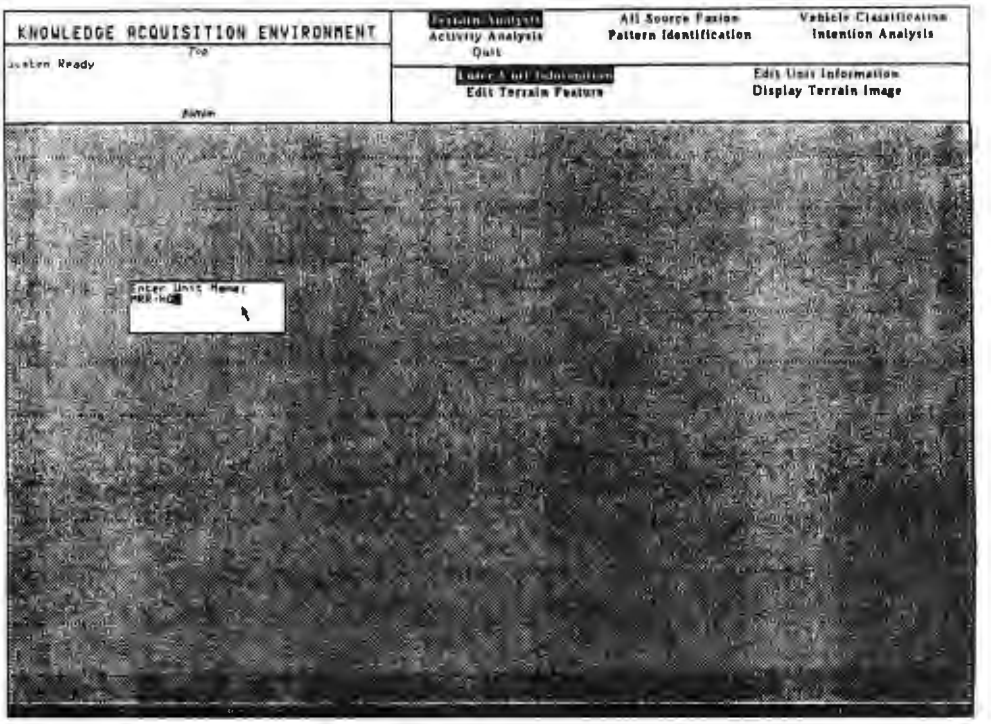


Figure 3-4: Entering Unit Name

features which are sufficient to describe the types of terrain suitable for a MRR HQ - Motorized Rifle Regiment Headquarters (Figure 3-5). From the choices made in the terrain feature menu, a window is configured to allow the user to enter the likelihood probabilities. As shown in Figure 3-6, each of the terrain features are expressed in finer detail in order to allow for an accurate specification. Each of these designations are fields in a terrain database in which each image is catalogued. The terrain feature names are mouse-sensitive to provide definition information (Figure 3-7). The sub-categories belonging to each terrain feature are also mouse-sensitive so that individual categories may be displayed on a map or image (Figure 3-8). This assists the domain experts in assigning the likelihood probabilities for a unit to be located in areas representing different types of terrain. Figure 3-9 shows the Canopy Closure (75 - 100%) feature displayed on a terrain area (map) under analysis. In this display, the darkened areas reflect the chosen terrain feature. The map is typically displayed on a color monitor where each terrain feature is assigned a color with color intensity reflecting the categories within each terrain feature. In this way, users may "overlay" different terrain features to aid in the analysis of the underlying terrain. Once the user is comfortable with the probabilities, the terrain analysis inference engine may be executed so that results may be obtained and the refinement process initiated. Changes and updates may also be saved, adding the MRR-HQ terrain information to the terrain knowledge base for later recall and editing.

Having completed an initial set of terrain analysis functions, development will now proceed to the spatial pattern identification domain. The spatial pattern identification domain model showed that users required a tool to enter spatial patterns as well as describe properties of the pattern. A property may be the number of objects, the distance between each of the objects or the type of each object in the pattern. The user can enter spatial information in two ways. The KAE will store a pre-defined set of patterns which the user may choose from in order to enter specific unit information or the user has the ability to add new patterns to

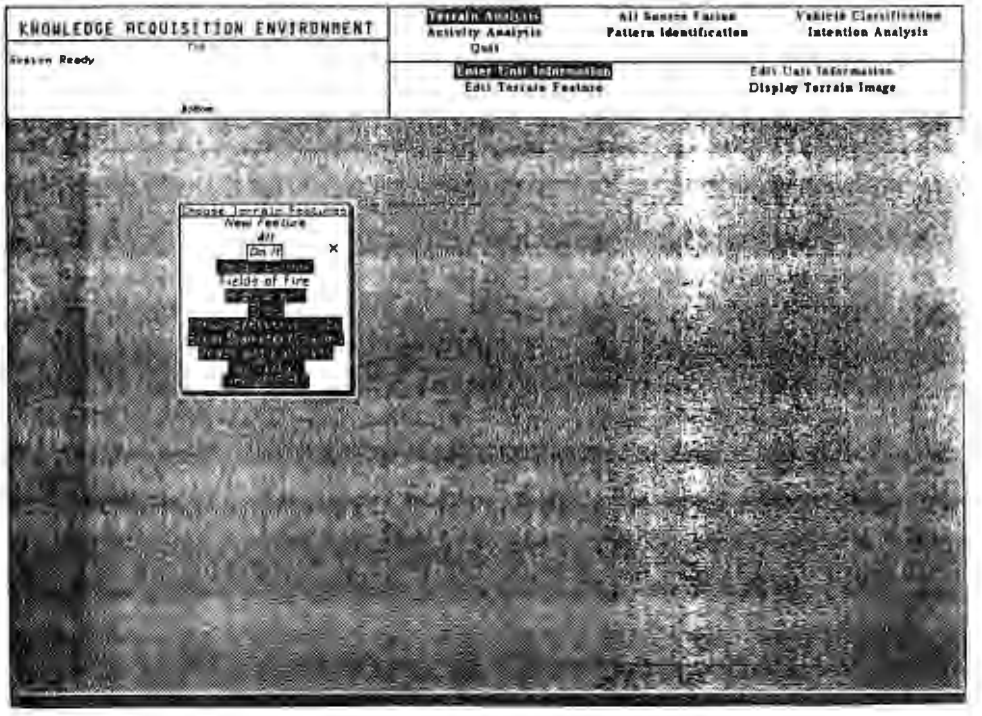
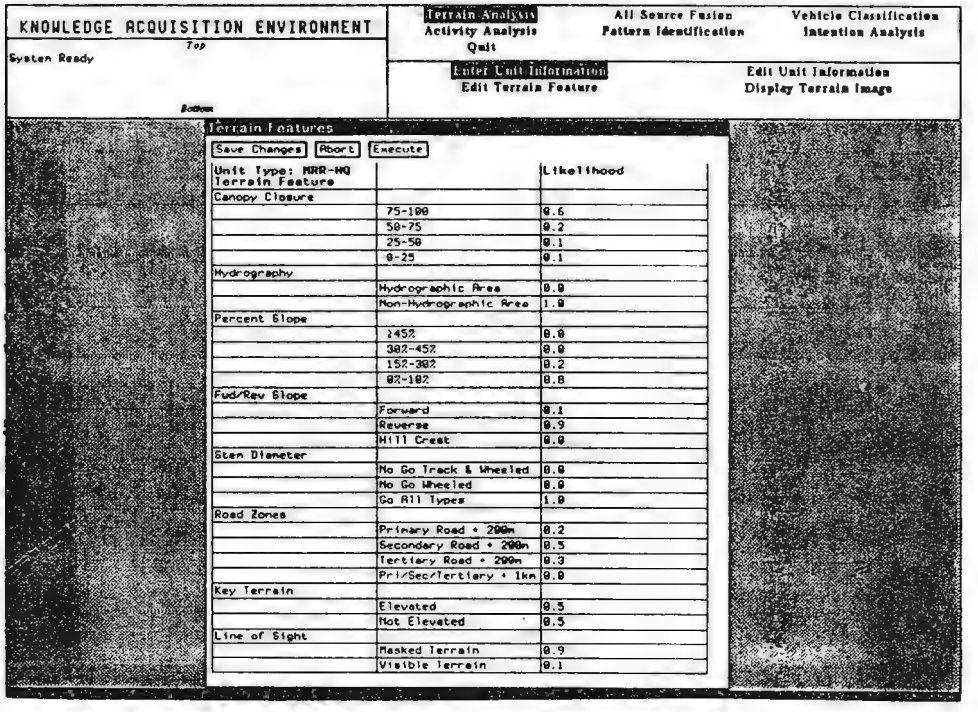


Figure 3-5: Terrain Feature Selection



Unit Type: MRR-HQ Terrain Feature		Likelihood
Canopy Closure		
	75-100	0.6
	50-75	0.2
	25-50	0.1
	0-25	0.1
Hydrography		
	Hydrographic Area	0.0
	Non-Hydrographic Area	1.0
Percent Slope		
	≥45%	0.0
	30%-45%	0.0
	15%-30%	0.2
	0%-10%	0.8
Fwd/Rev Slope		
	Forward	0.1
	Reverse	0.9
	Hill Crest	0.0
Stem Diameter		
	No Go Track & Wheeled	0.0
	No Go Wheeled	0.0
	Go All Types	1.0
Road Zones		
	Primary Road + 200m	0.2
	Secondary Road + 200m	0.5
	Tertiary Road + 200m	0.3
	Pri/Sec/Tertiary + 1km	0.0
Key Terrain		
	Elevated	0.5
	Not Elevated	0.5
Line of Sight		
	Masked Terrain	0.9
	Visible Terrain	0.1

Figure 3-6: MRR - HQ Terrain Feature Specification

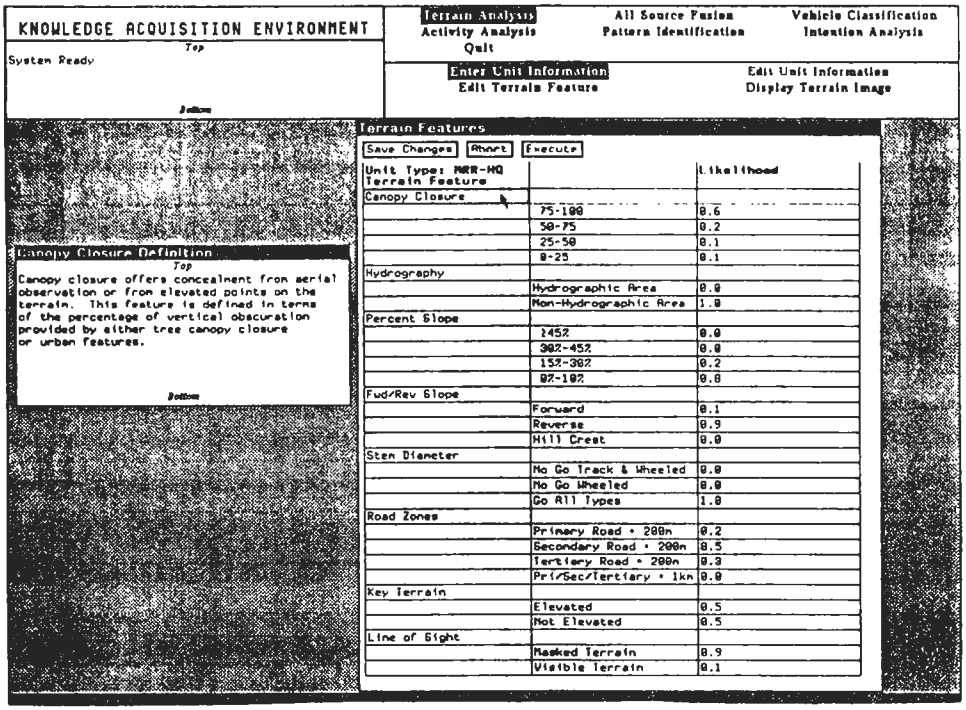


Figure 3-7: Terrain Feature Definition Information

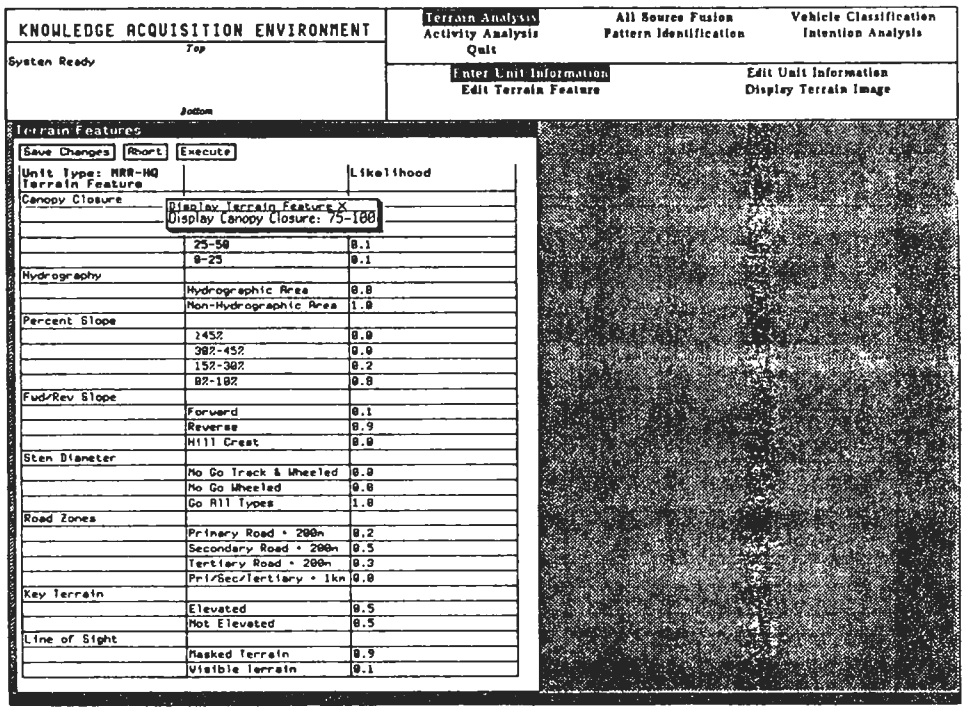


Figure 3-8: Selecting Terrain Feature to be Displayed

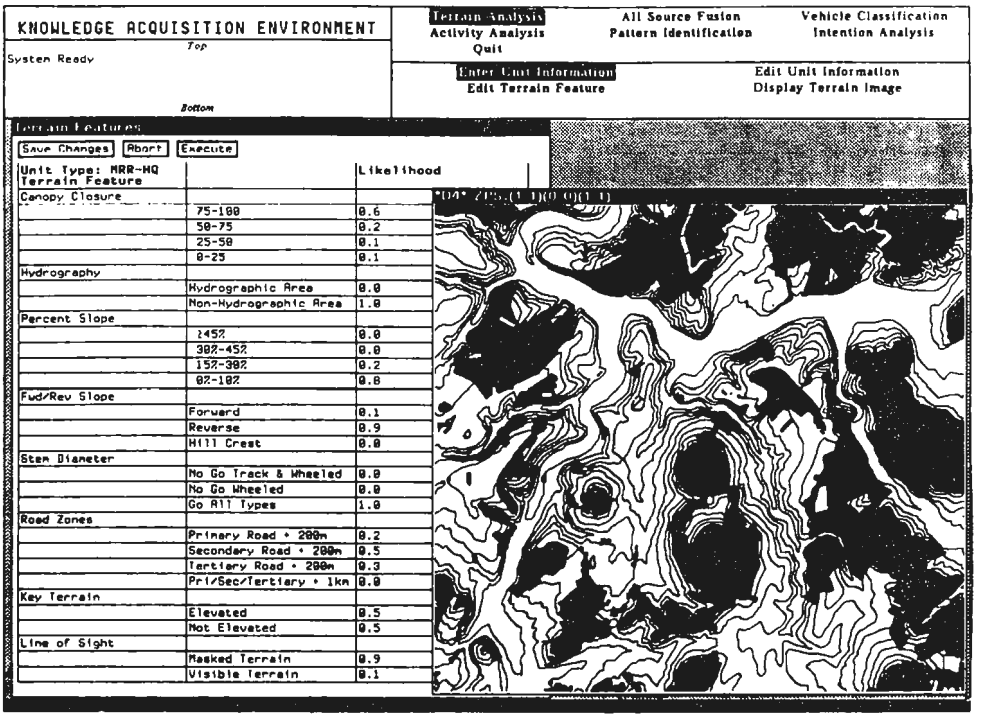


Figure 3-9: Canopy Closure Map Display

the KAE. The following paragraphs explain both modes of operation.

Figure 3-10 shows the spatial pattern identification sub-menu with "Define New Pattern" selected. The user is asked to give the pattern a name as well as specify the number of objects in the pattern. Figure 3-11 shows a graphic "working area" containing the specified number of objects (in no apparent order) along with a textual specification template. The user is prompted (in the notification window) to place the objects in the desired pattern. Each of the objects are mouse-sensitive and can be moved anywhere in the "working area". Once the user is comfortable with the layout the specification template can be "updated" to reflect the configured pattern (Figure 3-12). Notice the automatic addition of the angle descriptions. Finally, the user "saves" the pattern, causing the list of pre-defined pattern specifications to be updated reflecting the new triangle pattern. Editing can be accomplished form either the text or graphics display. Figure 3-12 shows changes in the graphic display reflected in the text specification. Figure 3-13 shows an angle change in the text display reflected graphically.

A second mode of spatial pattern identification involves utilizing one of the pre-defined patterns to describe the spatial layout of a specific unit (Figure 3-14). In this case, the newly created triangle pattern was chosen. The user is prompted to enter the unit which will utilize a triangle pattern - Motorized Rifle Regiment (MRR). Implicitly, the MRR inherits the properties of the triangle pattern (i.e., three objects in the pattern and the objects placed at the specified angles). The initial graphics display shown in Figure 3-15 is the triangle pattern that was previously created and save. The text display has additional fields allowing the user to specify ranges for angle and distance as well as indicate the type(s) of objects that compose the unit. The angle and distance fields can be edited. The user can change the graphic and cause the text display to be updated and vice-versa. The graphics window is initially configured with a system defined ruler measurement. The ruler can be adjusted to reflect any type of distance measure (i.e., miles,

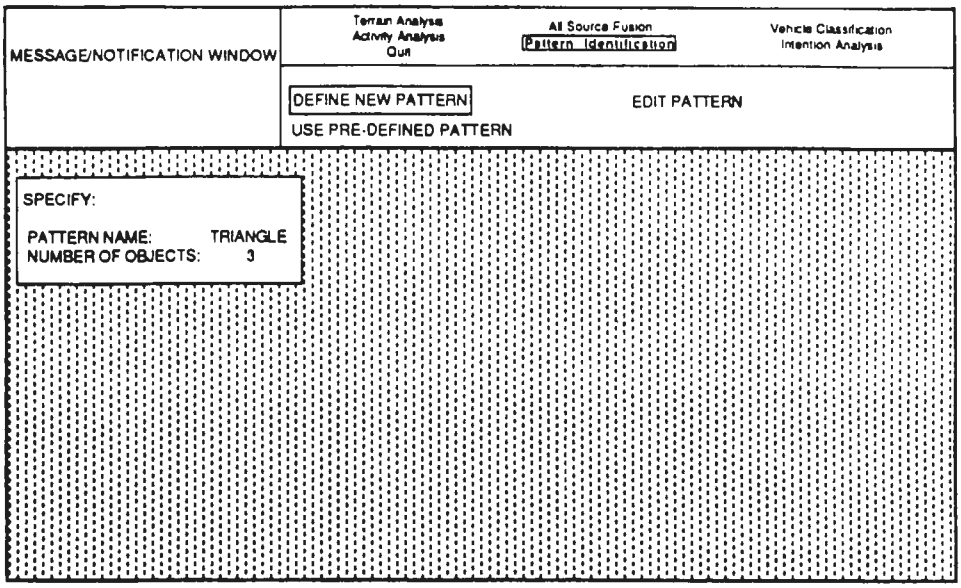


Figure 3-10: Spatial Pattern Specification

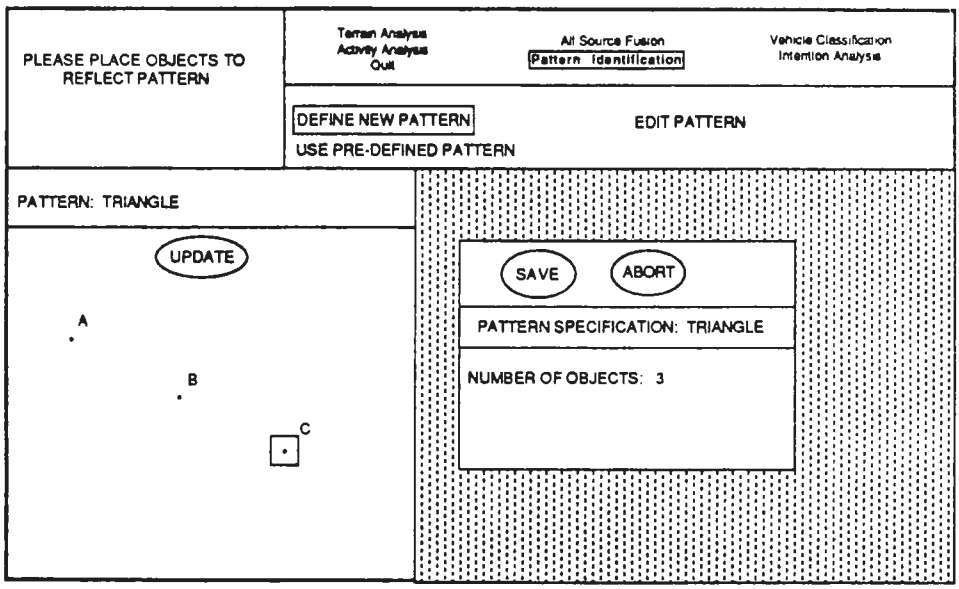


Figure 3-11: Triangle Pattern Specification

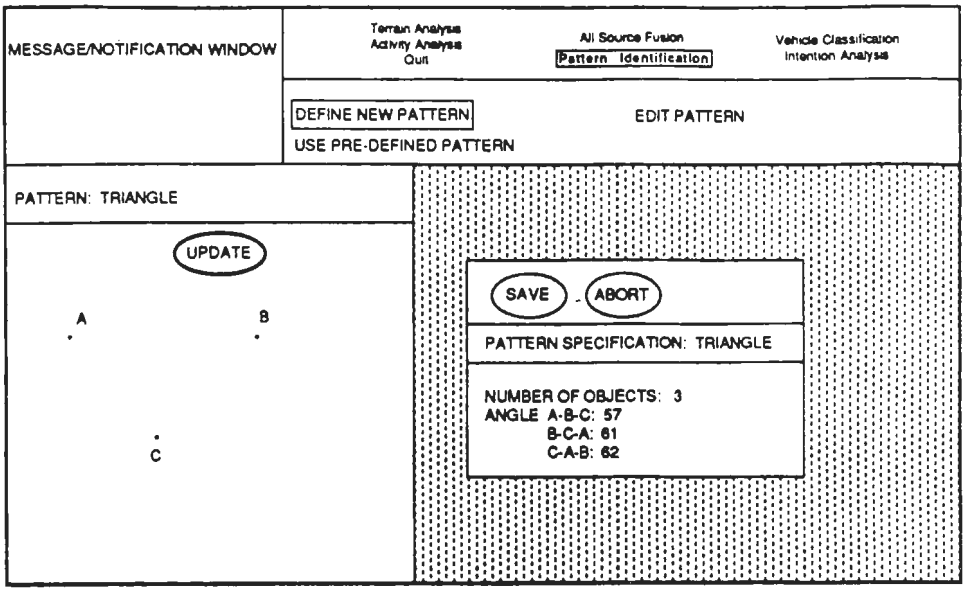


Figure 3-12: Spatial Pattern Updating

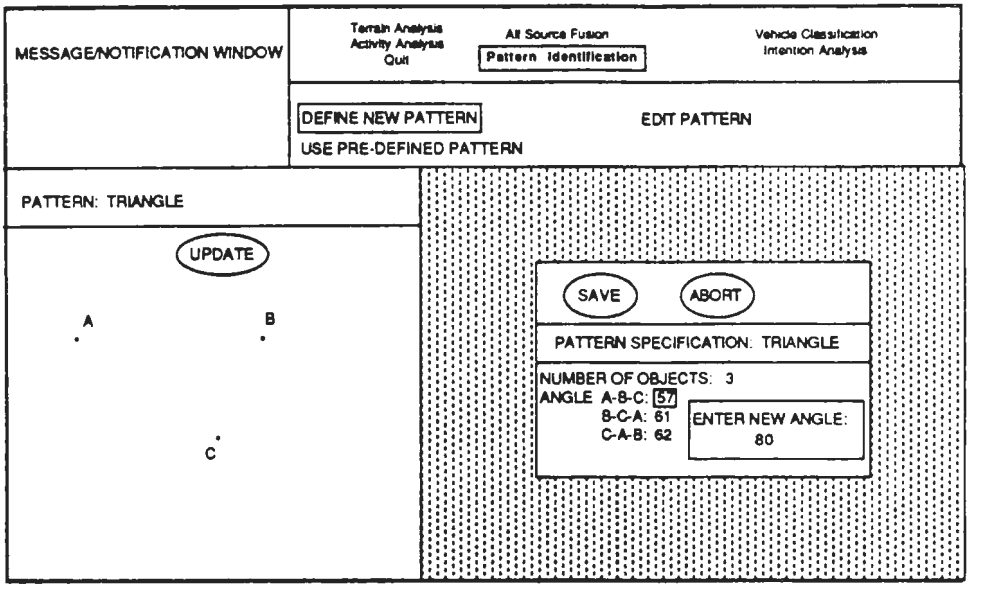


Figure 3-13: Spatial Pattern Modification

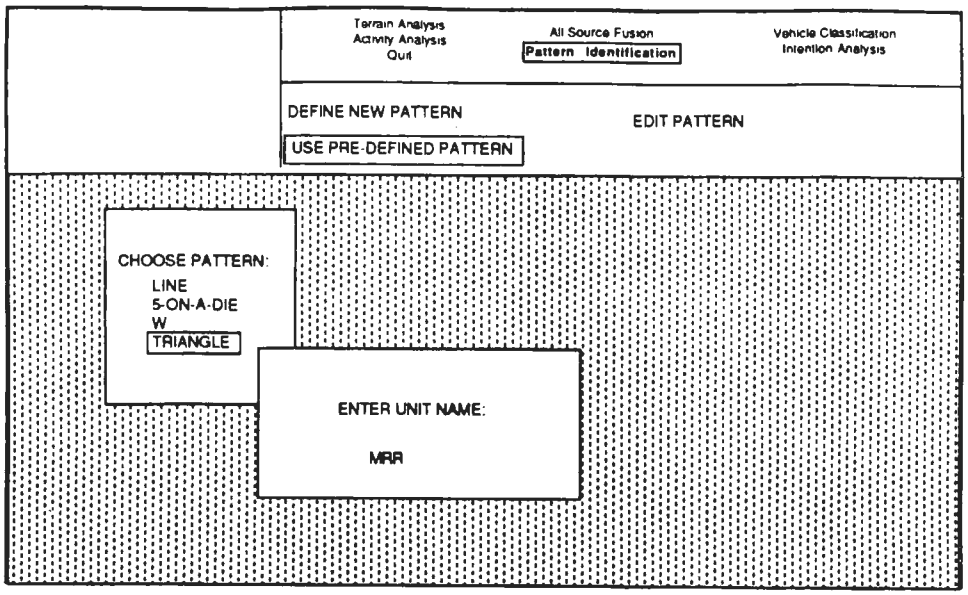


Figure 3-14: Spatial Pattern Specification - MRR

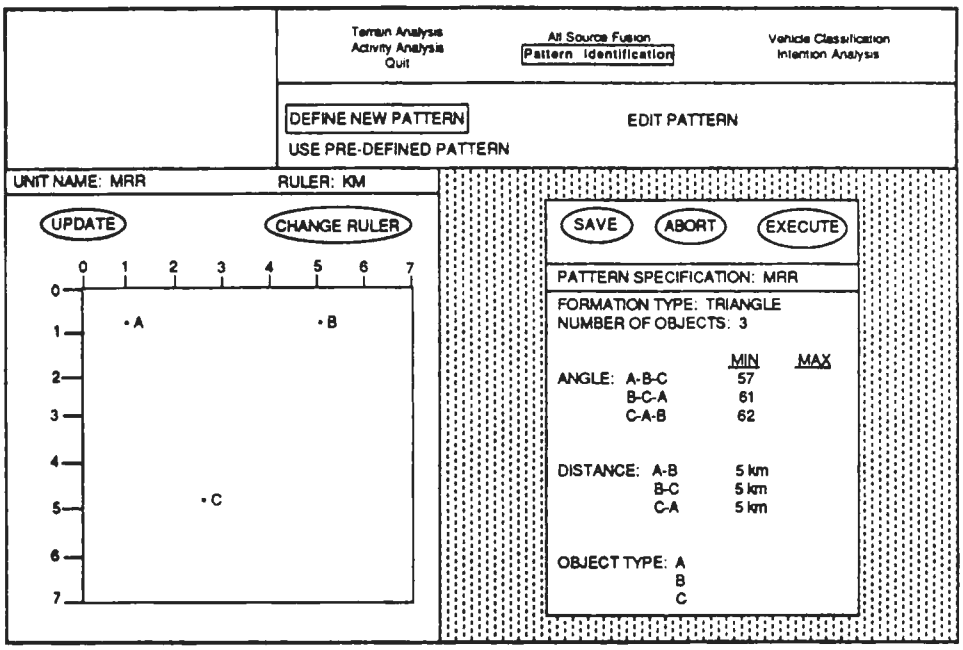


Figure 3-15: Initial Spatial Pattern Specification - MRR

kilometers, meters, etc.). In this example the ruler is shown in kilometers. Figure 3-16 show modifications occurring as a result of moving an object in the pattern. As soon as the 'update' request is received by the KAE the text display is changed. On the other hand, modifications can be made by 'mousing' on an entry in the text display, entering a new value and seeing the change in the graphic display. These features offer the domain expert the most in flexibility. Once the user is comfortable with the unit spatial description the text and graphic descriptions can be saved for later recall. Figure 3-17 shows the final result for an MRR.

The user interface and supporting tools described in the preceeding paragraphs constitute the initial development phase of the KAE. Domain experts and knowledge engineers can now begin to enter and edit terrain domain knowledge. Spatial pattern identification will be available soon. Minor changes to the current interface are anticipated but for the most part development will continue in the spatial pattern area.

3.2 TRANSLATION OF DOMAIN KNOWLEDGE

Within the KAE, raw domain information is translated into an intermediate representation (refer to Figure 3-1). This intermediate representation serves several functions. Initially, the intermediate representation acts as a buffer between the graphics and text entered through the user interface and the executable representation used by the application specific inference engine. Our goal is to define a rich set of intermediate representations through which the domain information may be translated (i.e., frames, rules, semantic networks, scripts). The intermediate representation chosen for any given domain will be influenced by the representation utilized by the application specific inference engines. Frames were chosen and implemented for the terrain analysis domain. Once the domain information is stored in the intermediate representation, another translation

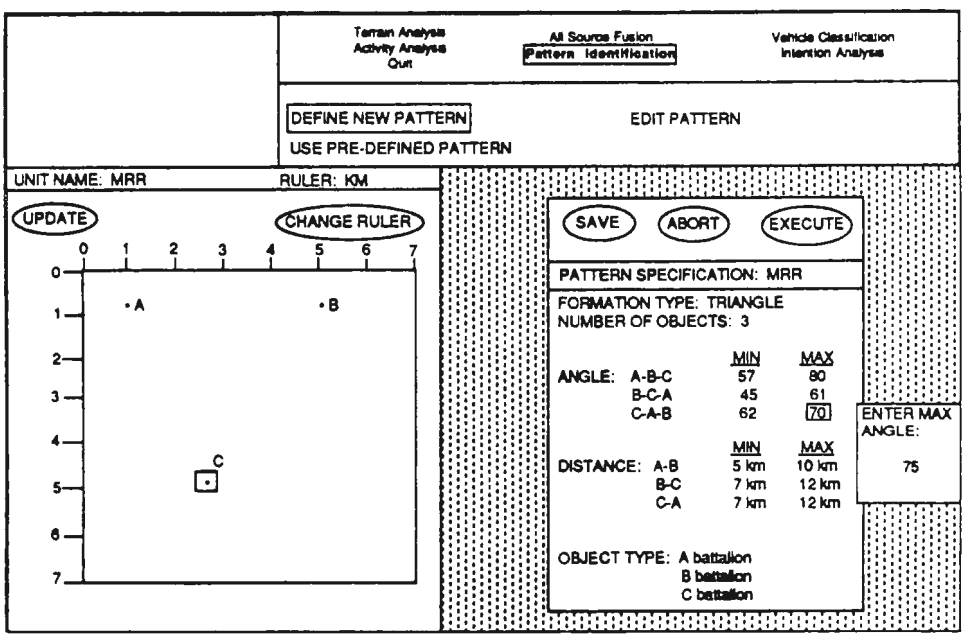


Figure 3-16: MRR Spatial Pattern Editing

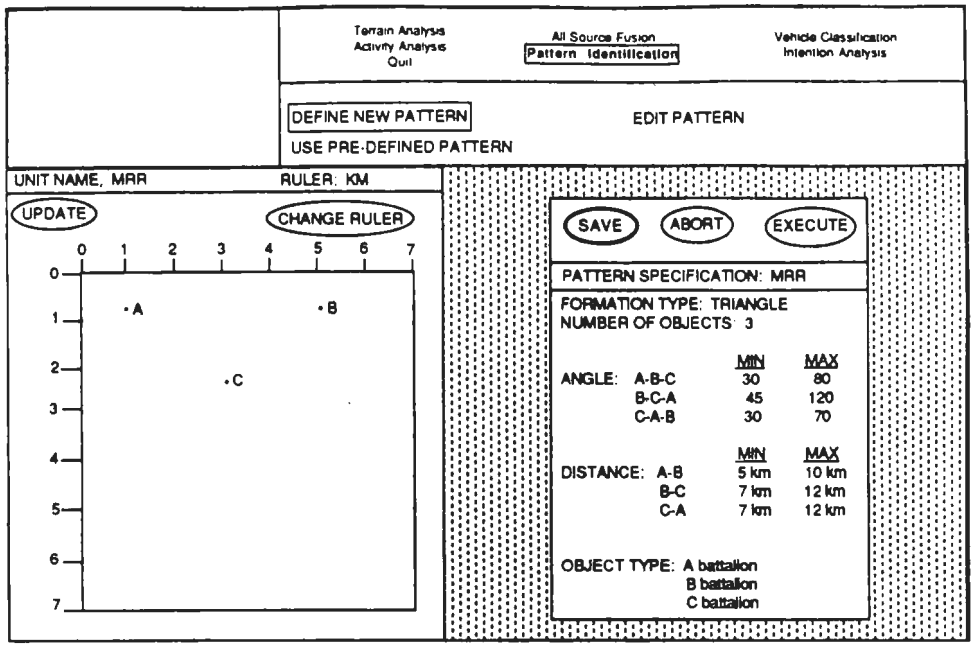


Figure 3-17: Final Spatial Pattern Specification - MRR

process transforms the intermediate representation into a form which can be executed. This translation process is only performed when the application specific inference engine is to be executed. This saves a significant amount of time during the initial entry and editing stages. The intermediate representation is also used as a means to transform results of the inferencing process into a form suitable for display to the user. Both the intermediate and executable knowledge representations are transparent to the user, freeing the expert from learning implementation details and allowing him to concentrate on the transfer of expertise.

As shown in Figure 3-1 a host of auxiliary tools (i.e., performance analysis and knowledge-base management) will also use the intermediate knowledge representation. These tools will be described in later sections.

3.3 EXECUTION AND ANALYSIS

A lot of current and previous effort is engaged in the development of a variety of application specific inference engines for scene analysis. Each of these will be utilized by the KAE. Knowledge bases for each supported application area will be submitted to the appropriate inference engine for execution. Once results are available, tools will be developed to assist in performance evaluation in the various application areas.

Most of the sample imagery used to develop knowledge-based image understanding and vision systems will have what is known as "ground truth" associated with the imagery. The ground truth describes the objects, patterns and activity contained in a particular image. A performance analysis tool will be used to compare ground truth information with inference results to test the overall accuracy of the inference process. This comparison can also guide the domain expert in the areas the knowledge base may be improved. For example, presenting the expert the

cases which did not produce useful results may show that a piece of imagery was bad, new terrain features need to be added to the terrain analysis area or certain objects are never found.

3.4 KNOWLEDGE-BASE MANAGEMENT TOOLS

As the KAE begins to store and manage large bodies of domain information knowledge-base management tools will be essential. These tools will use the intermediate representation (i.e., rules, frames, semantic networks). They will assist in both knowledge-base debugging and knowledge-base configuration management.

In the debugging area a tool should check for consistency in the saved knowledge base. As a knowledge-base grows, identifying the effects of modifying a concept or relation will be important. For example, a user decides to change the system definition within Terrain Analysis for "Hill Crest". We now need to be directed to all the units that rely on the hill crest feature and determine if the feature is still relevant to the unit and if the suitability measure should be modified. Another debugging aid would compute statistics based on inference results showing when a rule was last 'fired' or frame instantiated. Perhaps the domain information within the rule or frame is too special purpose, a particular unit is rarely seen or you are lacking imagery containing the unit. A graph could show a utilization ratio for each unit.

Knowledge-bases will change frequently. Any useful, nontrivial knowledge-base requires periodic correction, extension and improvement. In order to manage the updates and provide a historical summary of the knowledge-base development activity a configuration management tool will be embodied in the KAE. This tool will log who, when and prompt the user for why a knowledge-base was modified. "What" was changed should also be captured. This could be accomplished by recognizing the difference between the current and last version of the knowledge-base.

The set of tools described above is not a complete set. As domain experts begin to work with the KAE additional special purpose tools will most certainly be required.

4. CONCLUSION

As previously mentioned, a goal of the KAE is to maintain as much domain independence as possible and still be useful to a number of experts in different domains. We are hoping to gain insight into the amount of domain independence an effective KAE can possess.

Another area of study is the investigation of whether knowledge entered directly from a domain expert is "more valid" than that elicited by a knowledge engineer. Is there a difference? If so, what are the characteristics?

It is felt that with the basic KAE infrastructure in place, research and development can proceed in parallel. Domain experts can begin using the system allowing for active refinement and study into the areas cited above. Functionality will continue to be added to each of the core KAE components described. In addition, the basic structure will be in place to support new research in tool development, learning techniques, explanation and elicitation methodologies.

ACKNOWLEDGEMENTS

The author would like to thank R. Drazovich and C. McKee for their helpful comments and suggestions in the course of this research. C. Neveu implemented portions of the prototype version of the KAE.

Work for this paper has been sponsored by the Defense Advanced Research Projects Agency and the U.S. Army Engineer Topographic Laboratories under U.S. Army Contract #DACA76-86-C-0010.

REFERENCES

[Alexander et. al. - 86] - Alexander, James H., Michael J. Freiling, Sheryl J. Shulman, Jeffrey L. Staley, Steven Rehfuss, Steven L. Messick, "Knowledge Level Engineering Ontological Analysis", Proc. AAAI-86, Philadelphia, PA, August 1986.

[Boose - 86] - Boose, John H., Jeffrey M. Bradshaw, "Expertise Transfer and Complex Problems: Using AQUINAS as a Knowledge Acquisition Workbench for Knowledge-Based Systems", Submitted to the International Journal of Man-Machine Studies, December 1986.

[Boose - 84] - Boose, John H., "Personal Construct Theory and the Transfer of Human Expertise" Proceedings AAAI-84, Austin, Texas, August 1984.

[Buchanan - 85] - Buchanan, Bruce G., "Some Approaches to Knowledge Acquisition", STAN-CS-85-1076, Stanford University, Department of Computer Science, July 1985.

[Buchanan, Shortliffe - 85] - Buchanan, Bruce G., Edward H. Shortliffe, "Rule-Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project", Addison-Wesley, May 1985.

[DARPA, et. al. - 86] - DARPA, ETL, AFWAL, ASPO, "Advanced Digital Radar Imagery Exploitation System (ADRIES) Program Plan", October 1986.

[Froscher, Jacob - 85] - Froscher, Judith N., Robert J.K. Jacob, "Designing Expert Systems for Ease of Change", Proc. Expert Systems in Government Symposium, McLean, VA, October 1985.

[Gaines - 86] - Gaines, Brian R., "An Overview of Knowledge Acquisition and Transfer", Proc. Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, November 1986.

[Garg-Janardan, et. al. - 86] - Garg-Janardan, Chaya, Gavriel Salvendy, "A Conceptual Framework for Knowledge Elicitation" Proc. Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, November 1986.

[Kahn, et. al. - 85] - Kahn, Gary, Steven Nowlan, John McDermott, "Strategies for Knowledge Acquisition", IEEE Transactions on Pattern Analysis Machine Intelligence, Vol. 7, No. 5, September 1985.

[McKeown, Harvey - 87] - McKeown, David M. Jr., Wilson A Harvey, "Automating Knowledge Acquisition For Aerial Image Interpretation", CMU-CS-87-102, Carnegie-Mellon University, January 1987.

[Morik - 86] - Morik, Katharina, "Acquiring Domain Models", Proc. Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, November 1986.

[Musen - 86] - Musen, Mark A., Lawrence M. Fagan, David M. Combs, Edward H. Shortliffe, "Use of a Domain Model to Drive An Interactive Knowledge Editing Tool", MEMO KSL-86-24, Knowledge Systems Laboratory, Stanford University.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

USING CONCEPT LEARNING FOR KNOWLEDGE ACQUISITION

Ian H. Witten and Bruce A. MacDonald

Knowledge Sciences Institute, Department of Computer Science
The University of Calgary, 2500 University Drive NW, Calgary, Canada T2N 1N4

ABSTRACT

Although experts have difficulty formulating their knowledge explicitly as rules, they find it easy to demonstrate their expertise in specific situations. Schemes for learning concepts from examples offer the potential for domain experts to interact directly with machines to transfer knowledge. Concept learning methods divide into similarity-based, hierarchical, function induction, and explanation-based knowledge-intensive techniques. These are described, classified according to input and output representations, and related to knowledge acquisition for expert systems. Systems discussed include candidate elimination, version space, ID3, PRISM, MARVIN, NODDY, BACON, COPER, and LEX-II. Teaching requirements are also analyzed.

The difficult process of knowledge acquisition (KA) has emerged as a key technology underpinning the creation of expert systems. Unfortunately there is no all-encompassing, unified theory of how knowledge is acquired, and probably never will be. Consequently, practical KA systems must strive to provide a variety of different services, hoping to cover the requirements for many diverse domains of expertise.

Although experts have trouble formulating their knowledge explicitly as rules, they find it easy to demonstrate their expertise in specific situations. This paper reviews the potential contribution of published schemes for learning concepts from examples to knowledge acquisition for expert systems. The next section identifies a number of roles which learning might play in KA systems. Following that we discuss the meaning of concept learning, classify methods for doing it, and describe alternative representations for what is learned. The next sections present several schemes for learning, structured according to the classification (see Table 1). Then we consider the role of the teacher, for in the present primitive state of the art, concept learning systems need someone who not only has an analytical understanding of the problem domain, but also is acquainted with some of the internal workings of the system itself. Finally we discuss how each scheme meets the requirements for knowledge acquisition, and suggest areas where more work is needed to render the results of concept learning research accessible to practical KA systems.

Only techniques based on clearly defined algorithms are described here. In general, this means ones that we know have been replicated, in essence if not in detail, by researchers other than their inventors or colleagues. This eliminates a number of successful systems which have not been defined publicly in enough detail to be reproduced by others.

THE ROLE OF LEARNING IN KNOWLEDGE ACQUISITION

What would it be like to use a KA system that was capable of learning? The target expert system's knowledge base will contain facts, inference rules, and possibly some representation of the system's goals. Accordingly, there are a number of rather different opportunities for learning.

Acquiring facts

Rote learning of facts is a mundane requirement expected of all KA systems. The expert states facts explicitly and the system must store, index, and retrieve them as appropriate. For example, the user might say "Here are facts about fungal infections. There are three main types: a, b, and c. ...".

More challenging is the acquisition of procedures. The target system may be expected to execute various action sequences to acquire or process data relevant to the case in hand. The expert may prefer to communicate procedures by entering examples of their execution, instead of formulating an imperative program which covers all cases. By generalizing action sequences, the system is able to perform automatically tasks such as reading data from an instrument, formatting it, and activating appropriate analysis routines.

Learning rules

If the expert can articulate the general rules that embody his expertise, the process of knowledge transfer reduces to rote learning. For example, the expert might enter "All fungal infections are treated with drug p, q, or r; the explanation being that these drugs all create an environment inhospitable to fungi". However, it has long been recognized that experts find it very hard to formulate their rules explicitly.

Rules may be learned from selected test cases, if the expert can give examples which make the "how to" information clear. For instance, "This is an example of how to select a treatment for fungal infections of types a and b: skin infection symptoms are x, y and z, so you prescribe p and q". The more explicitly the expert can indicate the pertinent conditions surrounding an example, the less generalization is required of the system. Existing elicitation techniques give the expert the tools to study examples himself and extract the important factors.

Rules might be learned from hosts of examples. Many expert systems are designed to replace or augment a manual operation. It may be possible to extract rules by processing past cases and analyzing them for similarities. Such examples may be preselected by an expert to cover all major possibilities with little duplication. Alternatively they may constitute a random sampling of cases.

Active learners ask questions. The system might analyze the knowledge it has acquired, seeking alternative interpretations of cases it has encountered. When alternatives are found, it asks for more information, perhaps by creating a test case which discriminates between them and presenting it to the expert for judgement.

Learning what to do

While an expert system's goals are often implicit, or stored in the form of unanalyzable text strings, real experts understand their goals and can reason about them in a wider context. Moreover, they know their limitations, when to refer a case to another expert or seek a second opinion, and so on. The expert may be able to teach the system metaknowledge about its own goals and capabilities; for example, "Your aim is to take inputs of the type skin infection and produce outputs of the type treatment and explanation".

Learning from users

Real experts continually improve their performance on the job, by analyzing new situations brought to them for advice or judgement, and learning from them (Hawkins, 1983). They exploit these cases to acquire new knowledge, to increase their understanding of what they already know, and to operationalize existing knowledge from theory into rules which can be recalled rapidly in future. They are capable of "sustained learning", the integration of substantial new knowledge into a large established knowledge base on a continual, ongoing basis.

CONCEPT LEARNING

Current knowledge acquisition techniques take care of routine housekeeping, permit rote learning of whatever can be communicated explicitly, and are able to elicit relatively simple rules based on the attributes of example cases (eg Boose & Bradshaw, 1987). But they do not permit the inference of rules which operate on more powerful representations (eg nested structures), nor can they learn from cases presented to them by non-expert users. Methods of concept learning may be able to overcome these limitations, although the present state of the

art is primitive and offers ideas rather than well-developed algorithms to the KA toolbox. Concept learning systems take examples — which experts *can* provide — and create more general descriptions, often in the form of rules, which expert systems need.

What is meant by "concept learning"? Dictionary definitions of "concept" are remarkably vague, but have in common the abstract idea of a class of objects, particularly one derived from specific instances or occurrences. "Learning" also is a very broad term, and denotes the gaining of knowledge, skill, and understanding from instruction, experience, or reflection — in other words, "knowledge acquisition" by people. Notwithstanding the grand and sweeping nature of these ideas, the notion of concept learning as used in AI is much more constricted and specific. We will take it to denote *the acquisition of structural descriptions from examples of what is being described.*

Others have defined terms such as "generalization" (Schank *et al*, 1986), "inductive learning" (Michalski, 1983), and "inductive modeling" (Angluin & Smith, 1983) in almost identical ways. Not only are these all used to mean much the same thing, but what is learned is sometimes called a "generalization", a "description", a "concept", a "model", a "hypothesis". The field has not yet been successful in developing a satisfactory technical vocabulary; which term one favors seems purely a matter of taste.

Concept learning implies the acquisition of descriptions which make the structure of generalizations explicit. When a person learns a new concept, what she gains is knowledge that she can use in a rich variety of different ways: apply it to a particular case, reflect on it, tell it to a friend, criticize it, relate it to other ideas, and so on. For a person, learning is not simply a matter of acquiring a description, but involves taking something new and integrating it fully with existing thought processes. Existing programs which embody concept learning are not nearly so capable! However, they do acquire descriptions which are explicit in the sense that they can be communicated economically (for example, in the form of rules) and could plausibly support a variety of different kinds of reasoning. This orientation rules out, for example, connectionist models of learning, which embed knowledge in high-dimensional numerically-parametrized spaces, making learning into a process of weight adjustment. Since this does not result in concept descriptions that can be communicated economically or reasoned about, we will not discuss it further.

Mechanisms

There are almost as many paradigms for machine learning as there are systems. In fact, the variety of different, evocative, connotation-laden words used to describe simple mechanisms is one of the biggest problems in coming to grips with the field. Methods have been reported for learning by *analogy, being told, debugging, discovery, doing, examples, experimentation, exploration, imitation, instruction, observation, rote,* and *taking advice*. The apparent richness and variety of these approaches may give a misleading impression of a field teeming with fruitful techniques, with a selection of well-defined methods for tackling any given problem. Not so.

Here we distinguish the classes of concept learning mechanisms given below, and summarized in Table 1. While many taxonomies are possible, this one is intended to support distinctions that are particularly relevant to KA applications.

- *Parameter learning,* where the task is to optimize numerical parameters, often by detecting regularities in noisy situations. It assumes a particular description from the start, learning only the parameters of the model. Generally, large numbers of examples are used to allow incremental, hill-climbing style optimization and to overcome the effects of noise. We do not address parameter learning here.
- *Similarity-based learning,* where the space of concept descriptions is delineated in advance and searched for concepts which best characterize the structural similarities and/or differences between known examples.
- *Hierarchical learning,* where the concept description language is augmented as each concept is learned, so that new descriptions can build on old ones.
- *Function induction,* which constructs a function that accounts for a set of observed input-output tuples.
- *Knowledge-intensive learning,* which tackles the problem of relating new information to an existing body of knowledge. This area is as yet too undeveloped to attempt a taxonomy. Significant approaches are *explanation-based learning*, where a substantial chunk of information is learned by intensive analysis of just one example, and *learning by discovery*, where the system operates autonomously, performing

experiments to enhance its knowledge base.

Concept representation

Another important dimension of classification is how systems represent what they learn. Knowledge representation has always been a central topic in AI, and Barr & Feigenbaum (1982, pp.143-222) identify the representation schemes of *logic, procedural representations, semantic networks, production systems, direct (analogical) representations, semantic primitives,* and *frames and scripts.* It is worth pointing out that a data structure is not knowledge, any more than an encyclopaedia is. A representation of facts or rules only becomes knowledge when used by a program to behave in a knowledgeable way.

The requirement to acquire knowledge automatically from teacher or environment and integrate it with what is already known means that only the simplest representations are used by programs for machine learning. Separate representations are required for input examples and output concepts. Representations used for input include

- attribute vector (eg see Figure 1)
- combination of predicates (eg see Figure 2)
- nested term (eg see Figure 6)
- input-output tuples (eg see Figure 6);

and for concepts,

- vector of generalized attributes (eg see Figure 1)
- combination of predicates (eg see Figure 2)
- decision tree (eg see Figure 3)
- production rule (eg see Figures 3, 4)
- concept tree or semantic network (eg see Figure 5)
- recursive functional relationship (eg see Figure 6)
- nested numeric expression (eg see Figure 6)
- polynomial or other constrained numerical function (eg see Figure 6).

Table 1 shows the input and concept representations for the methods reviewed below.

Many learning methods apply to examples which can be expressed as vectors of attributes. The values an attribute can have may be *nominal, linear,* or *tree-structured* (Michalski, 1983). A *nominal* attribute is one whose values form a set with no further structure. A *linear* attribute is one whose values are totally ordered; ranges of values may be employed in descriptions. A *tree-structured* attribute is one whose values are ordered hierarchically. Only values associated with leaf nodes are observable in actual objects; descriptions, however, can employ internal node names where necessary. A nominal attribute is a special case of tree-structuring with no levels other than root and leaves; a linear attribute can be represented as a tree structure of intervals. Figure 1 shows an example of each kind, along with a sample object and a sample concept description which includes it.

Attribute vectors are not powerful enough to describe situations where each example comprises a "scene" containing several objects. The classic example is Winston's (1975) *arch* concept, defined as three blocks, two having the attributes required of columns and the third having those required of a lintel, with the columns set apart from each other and each supporting the lintel. Objects are characterized by their attributes. Moreover, pairwise (or more complex) relations may exist between objects. Such relations can be described by functions which, like attributes, may be nominal, linear, or tree-structured. For example, the result of a function *relative-position* which takes two objects as arguments could be a pair of linear values *x-distance, y-distance.* Alternatively it could be tree-structured, as illustrated in Figure 2. Objects and concepts in such domains are characterized by combinations of predicates.

Method	System	Input representation	Concept representation	Teacher
similarity based				
version space	(many)	attribute vector	vector of generalized attributes	supplies examples one at a time; results are insensitive to order, searching is not
	(many)	combination (invariably conjunction) of predicates	combination of predicates	
decision rules	ID3	attribute vector	decision tree	supplies examples all at once; examples may be noisy
	PRISM		production rules	
hierarchical	MARVIN	nested term	concept tree	supplies examples and classifies ones suggested by system
	ALVIN		semantic net	
function induction	MARVIN		recursive functional relationship	
	NODDY	input-output tuples	nested numeric expression	supplies examples one at a time
	BACON		product of linear and derivative operators	supplies table of experimental data
	COPER	input-output tuples + dimensions	polynomial of suitably transformed arguments	supplies initial data, performs experiments as suggested
knowledge-intensive				
explanation-based	LEX-II	conjunction of predicates	production rules	supplies examples one at a time
discovery-based	AM	domain simulation	frames containing rules	can suggest directions to explore

Table 1 Methods, systems, and data representation

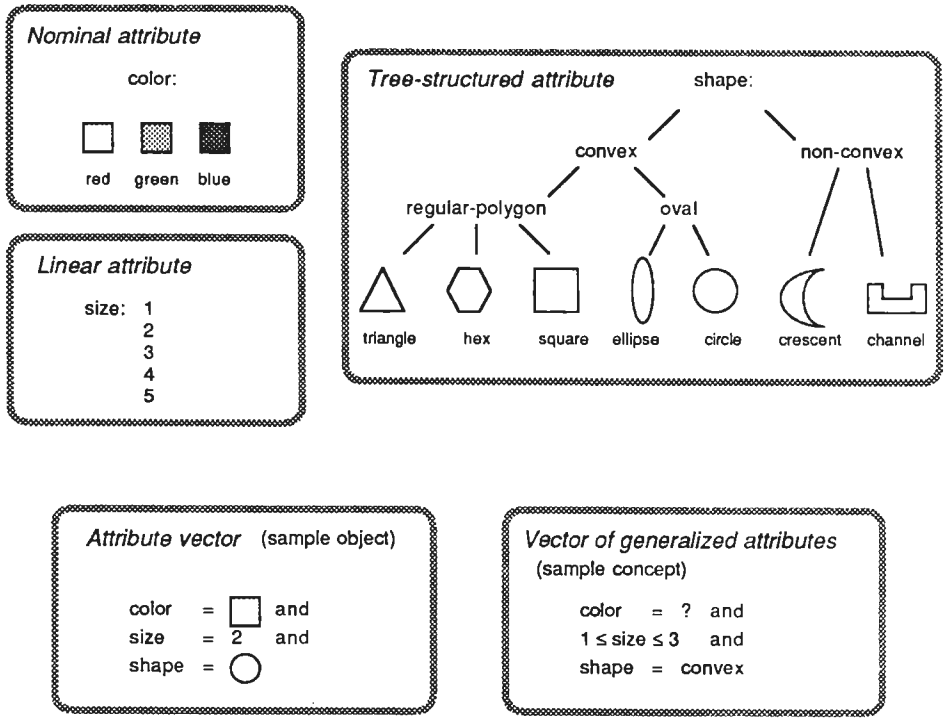


Figure 1 Attribute domain (adapted from Haussler, 1987b)

SIMILARITY-BASED LEARNING

Given a set of objects which represent examples and counterexamples of a concept, a similarity-based learner attempts to induce a generalized description that encompasses all the positive examples and none of the counterexamples. Interesting general issues include the conditions under which the procedure converges to a single description, whether the system can know that it has converged, whether the final concept may depend on the order of presentation of examples, and whether the training set is expected to be exhaustive (so that the concept description just discriminates between the examples presented) or representative (in which case some "generalization" is expected).

Candidate elimination and version spaces

A number of methods stem from the early experiments of Bruner *et al* (1956), who presented subjects with cards on which were drawn various figures differing in size, shape, color, etc, and indicated whether they were positive or negative examples of a concept. Attempts to inject some realism into this artificial task indicated that subjects behave differently when they can relate the various attributes to the real world. However, experimental studies continued to be preoccupied with information-processing aspects of the problem.

In situations where objects can be described by a finite vector of attributes, the inductive problem of creating an appropriate description can be transformed into a deductive one by circumscribing the language in which descriptions are expressed and searching for that description (or set of descriptions) which fits the examples given. The "version space" approach to concept learning (Mitchell, 1982) postulates a language in which objects are expressed, a language in which concept descriptions are expressed, a predicate which determines whether a particular object matches a particular description, and a partial ordering on descriptions (interpreted as generalization/specialization). Given a set of positive and negative examples of a target description, the problem is to find all descriptions that are consistent with the examples. This set is called the "version space".

In principle, the problem can be solved by enumerating candidate descriptions and striking out those which do not fit the examples that are presented. A positive example rules out all descriptions that it does not match, while a negative one eliminates those it does match. This is called the "candidate elimination" method. As each example is encountered, the set of remaining descriptions shrinks (or stays the same). If only one is left, it is the target description. If several are left, they may still be used to classify unknown objects. An unknown object which matches all remaining descriptions is included in the target; if it fails to match every one it is excluded by the target description. Only when it matches some descriptions but not others is there ambiguity; in this case if the classification of the object were known it would cause the version space to shrink.

Candidate elimination does not use the partial ordering on descriptions. The essential insight of the version space method is that the descriptions in the version space need not be stored explicitly, but can be inferred from its upper and lower boundaries in the partial order. The upper boundary comprises all members that are maximally general, the lower boundary all that are maximally specific. The boundaries can be updated incrementally as new examples are encountered. The upper boundary is initialized to comprise all maximally general descriptions (there may be several). In principle, the lower one could be initialized to comprise all maximally specific descriptions; since there may be a large number of these it is instead initialized to include the first example only (which must be positive).

Vectors of attributes. The candidate elimination and version space methods apply simply and directly when each object is described by an attribute vector, and the performance of version space algorithms in such domains has been studied extensively. It is well known that allowing disjunctions in the description language causes the version space to explode. Even with purely conjunctive concepts, however, if the number of attributes is large the size of one version space boundary can grow exponentially in the number of examples. This and other complexity results are discussed by Haussler (1987a).

Combinations of predicates. In structural domains, each example (or counter-example) comprises a "scene" containing several objects. Part of the problem in matching a scene with a structural description is determining an appropriate mapping between objects in the scene and those specified in the description. Figure 2 shows an example scene containing three particular objects — a, b and c. It also gives a description which refers to two variable objects x and y. Binding x to a and y to c shows that something matching the description is contained within the scene. But it is necessary to clarify how the description is to be interpreted. It might be

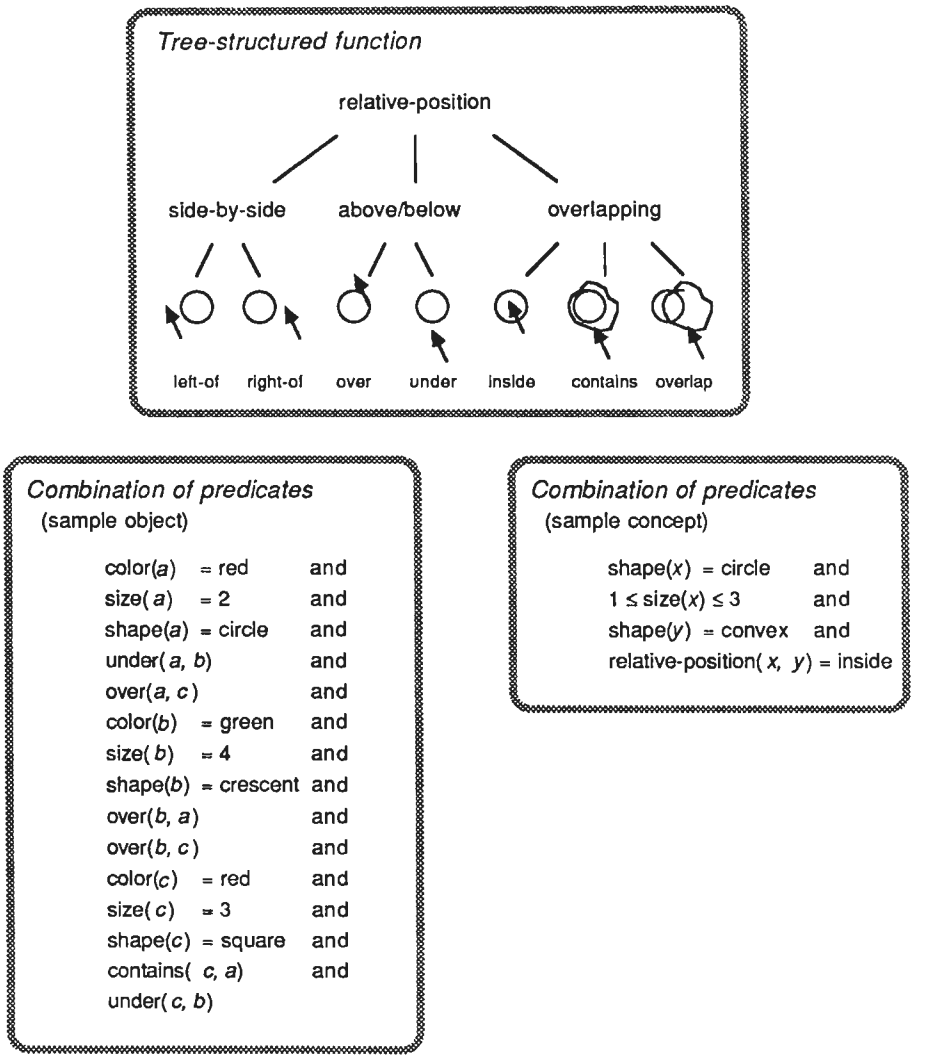


Figure 2 Structure domain (adapted from Haussler, 1987b)

prefaced by a statement such as

there exist different x, y with ...

giving what Stepp (1987) calls "contains" semantics. This is distinguished from "is" semantics, where the scene must have those parts mentioned in the description and only *allowable* extras. Under "is" semantics, optional parts (eg vines, spiders, dust, on an arch) would be observed in positive examples, and as more were seen the concepts used to describe optional parts would become more and more general. Negative examples containing ornaments would prohibit adornments of that kind.

Haussler (1987b) has obtained several theoretical results which indicate that, even with the simpler "contains" semantics, extreme computational complexity can be involved in working with version spaces of structured objects. The problems of testing whether one description is more general than another, and whether a description contains a given scene, are both NP-complete if we allow arbitrarily many objects in scenes and arbitrarily many variables in descriptions. Furthermore, the boundaries of the version space can become arbitrarily large. Even when each scene has exactly two objects, no binary relations are defined between the objects, and each object has only two-valued attributes, the size of boundaries can grow exponentially in the number of examples if there are sufficiently many attributes. Moreover, under these conditions it is NP-complete to determine if there exists any concept consistent with a given set of examples. These pessimistic results may indicate why even though Winston's early machine learning system used structured objects, relatively few subsequent projects have worked with structured examples (Stepp, 1987).

Creating decision rules

Another way to tackle the problem of concept learning is to use the training set to construct a decision tree or collection of rules which discriminates positive from negative examples. In the case of a tree, the root specifies an attribute to be selected and tested first, then depending on its value subordinate nodes dictate tests on further attributes. The leaves are marked to show the classifications of the objects they represent. For two-class problems these classifications are simply "positive" and "negative", but it is easy to accommodate situations where more than two classes must be distinguished. In the case of production rules, the training set is used to construct a set of rules which can be interpreted by an expert system in standard forward- or backward-chaining manner. In both cases, it is normally assumed that all examples are available and can be processed together to construct the tree or rules.

Decision trees. Procedures for top-down induction of decision trees were developed many years ago to model human performance in concept learning (Hunt, 1962; Hunt *et al*, 1966). It was found that subjects focus on positive instances, and they are assumed to build up a decision tree in order to categorize instances. These early programs sought an attribute which distinguished positive from negative instances; if there were none, they selected a frequent characteristic of the positive instances and made this the first test in the decision tree. The set of instances that passed the test were treated as a subproblem in exactly the same way.

The ID3 algorithm (Quinlan, 1986a) is a popular modern incarnation of these ideas. It does not attempt to model human performance but rather aims to minimize the number of nodes in the tree. It uses an information-theoretic heuristic to determine which attribute should be tested at each node, looking at all members of the training set which reach that node and selecting the attribute that most reduces the entropy of the positive/negative decision. By the nature of the algorithm, correct performance is guaranteed on the training set. Any "generalization" achieved depends purely on the representation of objects and the choice of training set. By seeking the simplest decision tree, ID3 finds the most economical way to classify the examples given.

Because of its simplicity and elegance, the ID3 procedure has been replicated and studied extensively. Having performed well in the domain of chess end-games (where the problem is more to summarize a large body of decisions economically than to generalize), it was soon adopted for a number of commercial applications. When presented with noisy data, the basic ID3 algorithm constructs huge decision trees which reflect the detail of every example seen. The resulting uneconomical representation is not a suitable vehicle for generalization, and several research efforts are underway to develop improved algorithms which copy with noise (eg Quinlan, 1986b; Niblett & Bratko, 1986).

An example of operation of ID3 (from Cendrowska, in press) concerns an adult spectacle wearer who consults an optician with a view to purchasing contact lenses, bringing along her spectacle prescription. We assume that, from the optician's point of view, this is a three-category problem with four factors *a*, *b*, *c*, and *d* to take into consideration. The complete decision table is given in Figure 3, and ID3 produces the tree shown.

Production rules. For incorporation into expert systems, the decision tree is usually converted into production rules such as those of Figure 3. These rules are supposed to represent the professional judgement of the optician, which he is likely to state in forms such as

> This patient is not suitable for contact lens wear because her tear production rate is reduced (rule 1: a=1 → decide 3).
>
> This patient can only be fitted with hard contact lenses because she is astigmatic. As she is young and has a normal tear production rate, hard lenses are not contraindicated (rule 6: b=1 & c=2 & d=2 → decide 1 and rule 7: a=1 & b=2 & c=2 & d=2 → decide 1).

However, Cendrowska (in press) has noticed that even when a decision tree has the minimal number of nodes, there may be redundancy in the rules extracted from it. One manifestation of this is that there is no single rule which corresponds to the second example above; rules 6 and 7 between them cover the situation. Because of the existence of rule 6, rule 7 could be generalized to

> rule 7': a=1 & c=2 & d=2 → decide 1

without affecting the outcome. The additional clauses in the ID3 rules could cause an expert system to call for redundant tests to be made, in this case by demanding a value for *b* even when *a*, *c*, and *d* are known and together determine the result.

Rather than attempting to clean up the production rules generated from the ID3 decision tree, it is possible to dispense with the tree altogether and generate rules directly. The PRISM procedure (Cendrowska, in press) generates those shown in Figure 4 for the example problem. Although the number of rules is the same as for ID3, they are simpler and more general — six have had redundant terms removed. On a larger problem with a training set of 647 examples, PRISM found 15 rules containing a total of 48 terms where ID3 found 52 rules containing 337 terms. Both sets correctly classified all members of the training set.

Limitations of similarity-based learning

Similarity-based methods are the most mature and immediately applicable for the purpose of knowledge acquisition. However, it has often been pointed out (eg see Schank *et al*, 1986, for a recent discussion) that these methods hardly relate to what people do in real life.

First, they use the wrong sorts of concept, representing an object like "chair" or "table" as a certain configuration of sticks and boards. Many everyday concepts do not even involve a common function, let alone a common appearance (Wittgenstein, 1958). Second, they embed examples in an artificial scenario where objects are lifted out of context, isolated, cleaned up, and presented individually to the system. Third, they assume the existence of a teacher who decides when to create a new concept and when to modify an old one. In real life, the environment neither identifies nor labels instances. Fourth, systems tend to embody the implicit assumption that categories are defined by a collection of features that are individually necessary and collectively sufficient. Real instances have indefinite numbers of features, some explicit, some inferred.

Attempts have been made to go some way towards meeting such objections by designing new learning mechanisms which are embedded in richer knowledge-based systems. These are discussed below under "Knowledge-intensive learning".

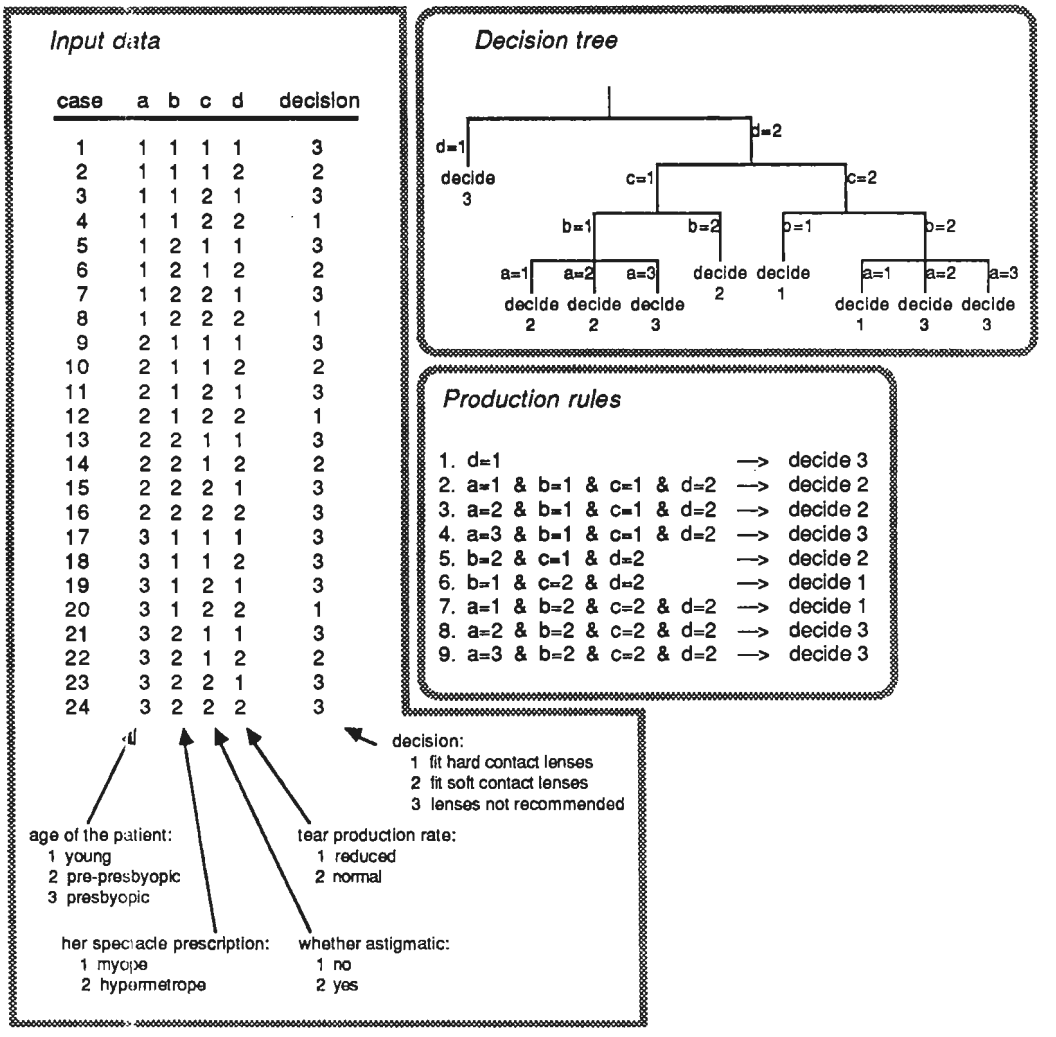


Figure 3 Decision tree and rules produced by ID3 (adapted from Cendrowska, in press)

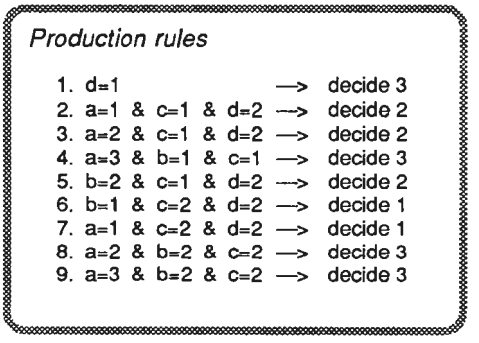


Figure 4 Rules produced by PRISM (adapted from Cendrowska, in press)

HIERARCHICAL LEARNING

It would be attractive if learning systems could build upon concepts already learned, and use them as components in newly-constructed descriptions. This might allow learning to be sustained over an extended period of time, instead of being done on a one-off basis. A mechanism for attempting this is reviewed below. However, it cannot support indefinitely sustained learning, for two reasons. First, the search space involved grows exponentially with the number of known concepts. Much more attention must be paid to the use of domain knowledge to curtail this growth before such systems become practical. Second, in order for learning to be truly sustained the teacher has to be able to correct misconceptions after they have occurred. Shapiro's (1983) techniques for debugging declarative programs may possibly provide a handle on this problem.

Examples which are presented to these systems may be atomic names rather than vectors of attributes or structures built from components. Concepts are stored in the form of a logic program. Each concept may be described by several clauses, which are interpreted as disjuncts. Each clause may have several terms which are interpreted as conjuncts. Terms are either primitives provided in the system, or other concepts; thus the structure is hierarchical.

Examples do not have to be atomic. They can have nested structure and, when used recursively, are capable of representing arbitrary list structures. If non-atomic examples are presented, they are decomposed to see if they can be expressed as compositions of other concepts. The process permits recursion, so that the concept of a number as a list of digits, as well as the procedure for appending such lists, can be taught by example.

Concept trees. A system called MARVIN learns hierarchical structures of concepts (Sammut & Banerji, 1983, 1986). Given an example, it searches for alternative ways to express it in terms of known concepts. Instead of awaiting further examples from the teacher to constrain its choice of expressions, however, it selects a tentative description, synthesizes a "crucial object" to determine if that description is a correct generalization, and asks whether it too is an example of the concept. If so, the tentative description is accepted as a valid generalization and attempts are made to generalize it further. If not, the tentative description is specialized to rule out the negative example and a new crucial example is synthesized. Thus a sequence of generalizations and specializations is made, each being tested by asking the teacher to classify a crucial example. When all possibilities for generalization have been exhausted, the description is stored and the teacher is asked for another example of the concept.

As an example, suppose MARVIN already has the concept *fish*, which includes *flying fish*, *cod* and *trout* as illustrated in the upper part of Figure 5. Upon learning that *trout* is a *swimmer*, it will hypothesize that all *fish* are *swimmers* and test this hypothesis by asking whether *cod* is a *swimmer*. Since it is, and no further generalizing hypotheses can be made, the system will record the fact that *fish* implies *swimmer* and await a new example from the teacher. Given *whale* as a new example, the concept is expanded by adding as a disjunct the fact that *whale* implies *swimmer*.

Semantic nets. The method implemented by MARVIN breaks down if the concepts are not structured and taught in the form of a tree. For example, the lower part of Figure 5 illustrates the *swimmer* concept in a richer, partially-ordered, knowledge base representing an animal kingdom. To create a strictly hierarchical structure it would be necessary to have used subclasses such as *swimming flightless bird* and *swimming aerial bird* when teaching the original *bird* concept. This is illustrated in the middle part of the Figure for a fragment of the animal kingdom. Notice that in the non-hierarchical representation on the right, each object can be represented by an attribute vector. Real-world knowledge bases are likely to fall between the extremes of hierarchical and completely flat representations.

It is desirable that new concepts (such as *swimmer*) can be added to general knowledge structures (like that illustrated at the bottom of Figure 5) without interfering in any way with previously learned concepts. This requires a more sophisticated method of selecting examples to be presented to the teacher. Given, for instance, that *man* is a *swimmer*, it is not immediately clear whether this is because it is a *mammal* and all *mammals* are *swimmers*, or because it is a *flyer* and all *flyers* are *swimmers*, or because it is both *mammal* and *flyer*, or purely by virtue of being a *man*. The question would be further complicated if there were superclasses of the *man* concept. Each hypothesis must be investigated. If a crucial object exists within the current tentative hypothesis but not within any competing ones, it is presented to the teacher. If such an object does not exist, the alternative hypotheses are investigated to discover the most general valid description (Krawchuk, 1987).

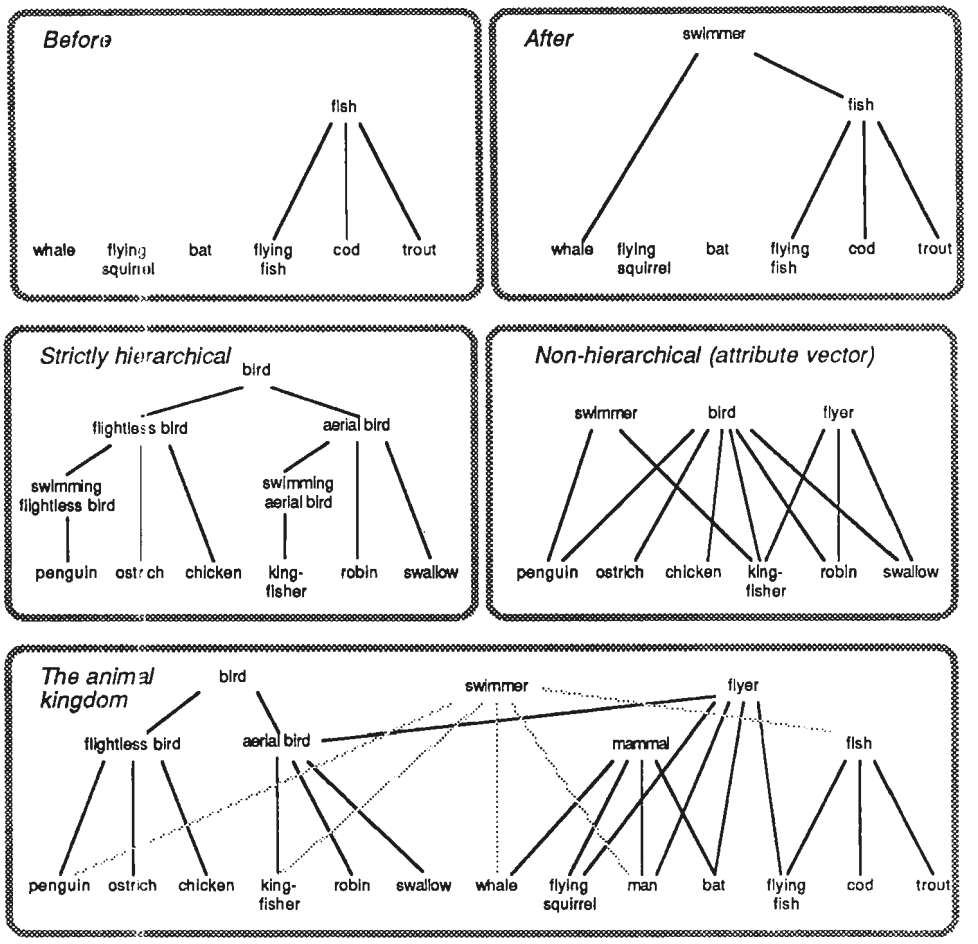


Figure 5 Teaching swimming in the animal kingdom

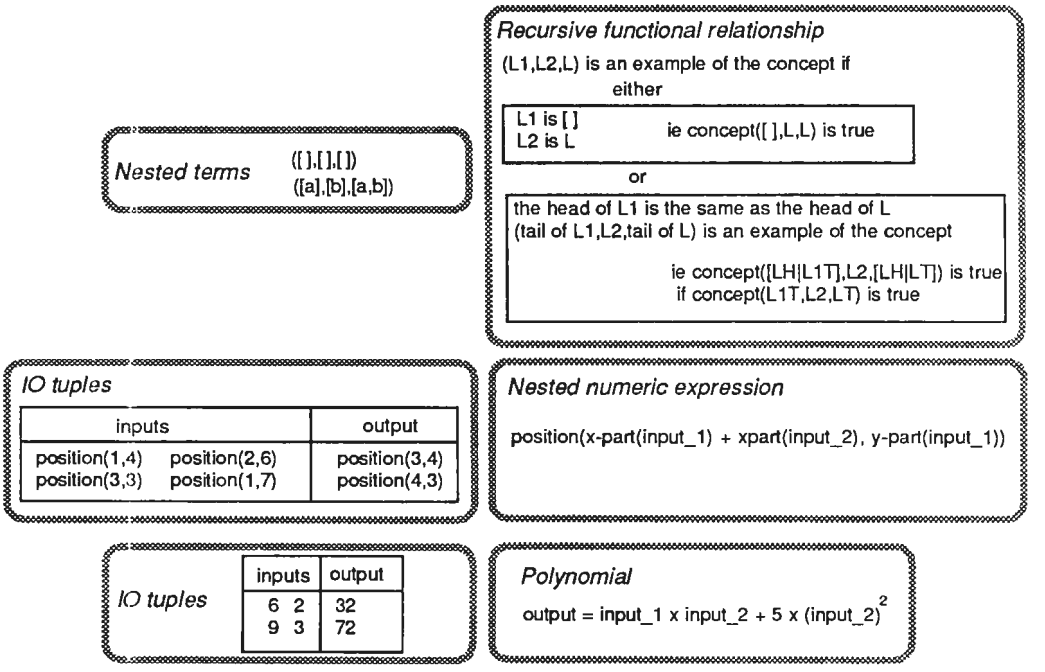


Figure 6 Examples of function induction

FUNCTION INDUCTION

Given a number of input-output (IO) tuples, function induction creates a function which computes the output of each pair from its input. In general this is very difficult, although for numerical functions whose form is known there is a well-developed body of theory in numerical analysis to calculate parameters of the relationship. Most artificial intelligence work on function induction concerns structured functions which are at least partially non-numeric. It is hard to identify common approaches, except to say that a massive search space is involved and searching is relatively unguided, since there is little potential for heuristically evaluating competing functional expressions. A selection of systems are reviewed briefly below.

MARVIN can be used for function induction as well as for hierarchical learning. Examples of functions which have been learned include

- list manipulation: appending, finding the maximum, and so on;
- arithmetic on numbers represented as strings of binary digits;
- simple insertion sort.

The first part of Figure 6 shows examples of the "append" concept, and the resulting function description. MARVIN proceeds by producing successively better approximations of the concept, oscillating between over-generalization and under-generalization until the target is finally reached. Each approximation is "tested" by synthesizing a new object and asking the teacher if it is an example of the concept. MARVIN's greatest weakness is that it cannot invent new existentially quantified variables. This eliminates the possibility of learning many interesting procedures (including, for example, more efficient sorting algorithms such as quicksort).

Function induction is employed as a component of NODDY, a system for learning robot programs from traces of their execution (Andreae, 1984a, 1984b). When presented with a list of IO pairs gleaned from successive incarnations of the same robot procedure, and a list of known constants that can be used, it attempts to induce the IO function. The partially complete function is represented as an expression which contains a "gap". This expression is initialized to the identity function with the gap standing for the unknown function to be induced. A list of candidate operators is programmed in, and the gap is closed if a single operator is found which generates each output from the corresponding input. Moreover, if there is an operator which maps all IO pairs to the same constant, the gap can be closed by using that operator's inverse. Otherwise, a new set of expressions is constructed, each of which modify the gap by adding an operator to it. Along with each new expression, a new set of IO pairs is constructed by applying the new operator to the input values. After pruning duplicate expressions and expressions corresponding to inconsistent IO pairs, the algorithm repeats.

It proves surprisingly difficult to characterize the functions that this method can induce. They are of course limited to combinations of the operators provided. The algorithm is constrained to introduce no more than one new constant, to avoid searching the infinitude of functions like $f(x) = x+a+b$ where $a+b=c$. (Actually, functions may have several inputs, and a separate constant can be introduced for each input variable.) Although the search for functions is strongly constrained by typing variables and operator arguments, it is nevertheless still an incredibly expensive search, and NODDY attempts function induction only when other evidence suggests that a functional relation may exist, and simpler methods fail. The middle part of Figure 6 shows example IO tuples and the nested expression that NODDY generates.

One AI system which works on wholly numeric functions is BACON, which attempts to discover empirical scientific laws by inducing functions that account for observed data (Langley *et al*, 1983). It is given as input a table of values of dependent and independent variables, and looks for monotonic relations between them. If one is found, it is tested to see if it is linear. If not, the slope of the curve relating the two variables is calculated and added to the table as a new variable. The lower part of Figure 6 illustrates the input and concept descriptions. From appropriate tables of data, the system discovered various relations in physics (eg Snell's law, conservation of momentum, gravitation); adding a heuristic which finds common divisors allowed it to interpret correctly data on the results of various chemical reactions. More recent work on the system is aimed at making it able to deal with noisy data using a hill-climbing strategy (Langley *et al*, 1986).

A more structured approach to deducing physical laws is to use dimensional analysis in concert with observed data. Kokar's (1986) system COPER works with physical quantities, for instance 3 m, 5 kg/m^2, 7 s. It takes a set of numeric observations, along with the dimensions of the variables involved (eg $m^1kg^1s^{-2}$ for force). It is also

told which are dependent and which are independent variables. When given data representing the observations in appropriate experiments, the system uses dimensional analysis to tell whether the independent variables are complete (ie sufficien to determine the dependent variable's value uniquely). For example, given a set of data representing speed, time, and distance traveled by a falling body, the system can detect that a variable (g, the acceleration due to gravity) has been omitted. It then suggests experiments from which it can infer the dimensions of the missing variable (m/s^2).

Having determined the necessary arguments, COPER goes on to discover the exact functional relationship by searching the space of polynomial functions. Any function (fulfilling some formal conditions) can be approximated to arbitrary accuracy by a polynomial. However, alternative sets of arguments can be used for the functional relationship, and if the wrong set is chosen this may lead to an artificially complex form. For example, suppose area (m^2), time (s) and mass (kg) are the arguments. Even though complete they may result in an overly complex polynomial, for it may be necessary to approximate a square root to derive distance (m). Consequently after searching polynomials up to a certain complexity the system selects alternative bases and tries those instead. If none gives a satisfactory result (ie a good approximation), each base is reconsidered using more complex polynomials.

KNOWLEDGE-INTENSIVE LEARNING: EXPLANATION BASED

It is often argued that an explanation-driven process is essential for reasonable generalization. Correlations derived purely from empirical observations are much less convincing than a theory which explains them. Explanation-based generalization methods take a single example and deduce a general rule by relating it to an existing theory. They do not learn the theory. Indeed, since the general rule is already implicit in the theory, they learn nothing new. What they do is use examples to guide the operationalization of knowledge already implicitly known, so that it can henceforth be employed more efficiently. The knowledge operationalized pertains to the original example, and probably to many others.

The earliest example of explanation-based learning is LEX-II, which learns problem-solving heuristics in the domain of symbolic integration. It starts with a set of operators for performing integration, corresponding to a table of integrals and a set of techniques for integration (eg integrating by parts). With each operator are specified preconditions which indicate when it can validly be applied. Solving a problem consists of applying a series of operators to derive an equivalent expression containing no integrals. The heuristics that LEX learns indicate when it is useful to apply the operator to solve an integration problem. Given an example expression to integrate, LEX embarks on an extensive search to obtain an appropriate sequence of operators. It then traces the example back through the sequence, at each stage using the preconditions to generalize the expression as much as possible consistent with that operator still being applicable. The result is an integration sequence for a more general expression than the original example. Finally, the operators used are stored along with the generalized expression to which they apply, so that they can be quickly chosen if in future a matching expression is encountered.

Figure 7 illustrates explanation-based learning in a blocks world (Mitchell *et al*, 1986). The goal concept is *safe-to-stack*, where

> it is safe to stack object x on object y if y is not *fragile* or x is *lighter* than y,

An example is given which says that it is safe to stack a particular box (whose color, density, volume are given as attributes) on a particular endtable. According to the domain theory, which is known in advance, endtables have a weight of 5 and can be deduced to be fragile. Moreover, the theory shows how to calculate weight from volume and volume, and how *lighter* is defined. The goal concept cannot be applied directly because *fragile* and *lighter* are not "operational" concepts; that is, they are not given directly but must be derived from the attributes of the objects through a proof procedure. Once a proof has been found for this particular example, however, the concepts involved can be generalized to produce the rule

> it is safe to stack object x on an *endtable* provided its *volume* times its *density* does not exceed 5.

Notice that the rule suppresses irrelevant details of the example (box, color) and identifies the relevant constraints on the other attributes (weight, density). It can be used to test rapidly whether objects are stackable

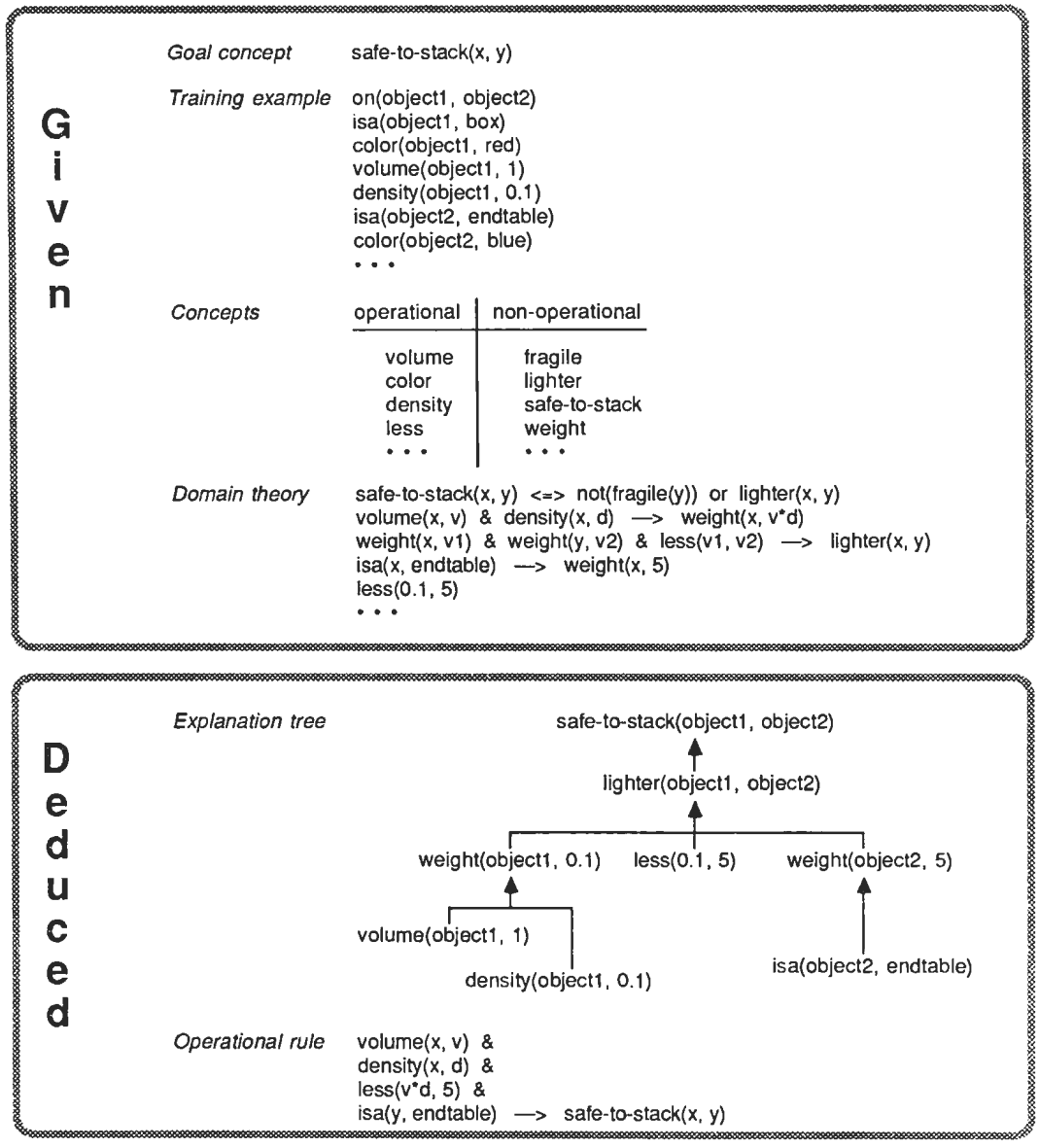


Figure 7 Explanation-based generalization
(adapted from Mitchell et al, 1987)

on endtables.

Explanation-based generalization is a two-step process, first creating an explanation that distinguishes the relevant features of the examples from the irrelevant ones; and then analyzing it to determine the minimum constraints that are sufficient for the explanation to apply in general. Because the explanation is created for the given training example, it does not necessarily explain every possible example of the goal concept. The final concept is typically a specialization of the goal concept rather than a direct re-expression of it.

A number of extensions to the view of explanation-based learning presented here have been identified (DeJong and Mooney, 1986). First, the proof on which generalization is based need not be discovered by the unaided system; any hints which a teacher might be able to provide can be taken into account, giving the system the character of an apprentice. Second, concept learning often involves specialization as well as generalization, and the explanation-based approach can be used for this too. Third, in more complex domains it becomes necessary to consider how predicates might be generalized; otherwise spurious conditions could become embedded in the generalized rule. Finally, the method described operates only on attributes and should be extended to structured objects.

Explanation-based learning usually presumes the existence of a strong domain theory in which proofs can be constructed that show why a particular example is valid. Some experiments have been reported with a kind of explanation-based learning supported by a very weak theory (Pazzani, 1987a). The domain chosen is physical causality for everyday events, and rules include

> if an action on an object always precedes a state change for the object, then it may cause the state change;
>
> if two similar actions performed on an object by different actors have different results, the differing features of the actors may be responsible for the different results.

Such rules are used to construct "explanations" for example events; competing explanations are resolved by a preference ordering or the rules. The explanation is used to choose which features of the events should be held accountable for their classifications. Of course, subsequent examples may contradict the reasoning, in which case an alternative explanation is constructed using lower-priority rules.

KNOWLEDGE-INTENSIVE LEARNING: DISCOVERY BASED

Explanation-based learning only addresses part of the objections to similarity-based learning which were raised earlier. Isolated attempts have been made to meet what is perhaps the major shortcoming, that examples are lifted out of the world, cleaned up, and presented to the system by a teacher who makes all the important decisions about when to create new concepts. Still the most impressive are the AM and EURISKO systems of Lenat (Lenat, 1978, 1983; see in particular Lenat & Brown, 1984) which work on their own to discover interesting concepts in elementary mathematics and interesting heuristics. However, research in this area is quite immature and does not yet form a foundation for practical KA techniques.

TEACHING REQUIREMENTS

A concept learning system's practical utility obviously depends critically on its teaching requirements. Teaching differs from programming in that a teacher does not use a formal model of the learner whereas a programmer normally expects to know exactly how his instructions are going to be interpreted. In other words, a teacher adopts the *intentional stance* while a programmer adopts the *design stance* (Dennett, 1981). Ideally, therefore, to be a good teacher one need know nothing about the internal structure of the learner, nor about the representation it uses for concepts.

Moreover, experts who use concept learning for knowledge acquisition are unlikely to have a clear analytical understanding of the task domain — otherwise more explicit methods of knowledge transfer will probably prove more appropriate. In general, people find it difficult to translate their own expertise into explicit descriptions. Consequently concept learning systems should be able to work with the kind of examples that a teacher finds it natural to provide, ordered in a way that is natural to him.

The teaching requirements and conditions under which learning is to occur provide a useful dimension for classifying learning mechanisms. Anything that is learned must be communicated through the expert- or user-machine interface. For example:

- *How cooperative/skillful is the teacher?* This dimension ranges from no teacher at all (input coming raw from the environment) through a naive user, a domain expert, to a teacher who is prepared to program in a conventional programming language and thereby circumvent the need for learning. Must the teacher show all working, give "near-miss" examples, teach simple concepts before complex ones, and so on (Witten & MacDonald, 1987)?
- *How reliable are the examples?* The examples provided may be noisy or exact. Much work in concept learning assumes exact data so that structural regularities can be learned without the problems of statistical detection.
- *Is learning done incrementally or retrospectively?* It may be that the current description must be augmented as each example is encountered; alternatively it might be possible to process all examples together. Batch operation is infeasible for systems which attempt sustained learning. In the incremental case, is the final concept supposed to be independent of the order of examples?
- *How much knowledge must the system have already?* There is an oft-quoted aphorism that in order to learn something, you must nearly know it already. The amount of knowledge that a system already possesses about the problem domain critically affects the kind of things it can be expected to learn. Even similarity-based techniques embody substantial *a priori* knowledge in the form of languages in which examples and concepts are expressed.

Table 1 provides some indication of the teaching requirements for each method discussed.

Formal aspects

A basic theoretical distinction has been shown between presenting positive examples only, presenting both positive and negative examples, and allowing the learning system to choose examples itself and have them classified by an informant. In the first case, it is assumed that all examples are included eventually. In the second, it is assumed that every member of some universe which contains all possible descriptions is shown eventually; this allows an informant to be simulated by simply waiting until a selected example occurs in the presentation sequence. Under these conditions, Gold (1967) proved formally what one expects intuitively, namely that the second and third methods are more powerful than the first. They permit primitive recursive languages, which include the context sensitive, context free, and regular languages, to be learned eventually, whereas with positive presentations only, the inclusion of even one infinite language in the set of potential concepts can destroy learnability.

There is a curious theoretical result which relates to the sequence of examples presented. Gold (1967) showed that descriptions unlearnable from positive and negative examples *can* be learned under a special condition: the presentation sequence must be sufficiently regular. Moreover, positive examples alone suffice! For instance, descriptions from the class of recursively enumerable languages (which include the primitive recursive ones mentioned above, and more besides) are not normally identifiable even from positive and negative examples, nor indeed using a helpful informant who is prepared to classify arbitrary examples. But a suitable, highly contrived, presentation sequence of positive examples can render such descriptions learnable. In essence the learner can identify the algorithm used to select the examples, thereby identifying the set from which they are drawn. Here the importance of the teacher's selected example sequence is clearly reflected at the theoretical level. A good sequence of examples does more than speed the search, it makes learnable the otherwise unlearnable.

Felicity conditions

A skilled teacher will select illuminating examples himself and thereby simplify the learner's task. The benefits of carefully constructed examples were appreciated in the earliest research efforts in concept learning. Winston (1975) showed how "near misses" — constructs which differ in just one crucial respect from examples of a concept being taught — could radically reduce the search required for generalization. Confident that its teacher is selecting examples helpfully, a learning system can assume that any difference between an example being shown and its nascent concept is in fact a critical feature.

The notion of a sympathetic teacher has been formalized in terms of "felicity conditions", constraints imposed on or satisfied by a teacher that make learning better than from random examples (Van Lehn, 1983). One obvious condition is that the teacher should classify examples correctly, and not (intentionally or unintentionally) mislead the student. Another is that if the absence of something is important, the teacher should point it out explicitly. If in the classic example of the concept of "arch" it is necessary that a gap be left between the pillars, the gap should be explicitly labeled in examples. More generally, the teacher should show all work and avoid glossing over intermediate results. Examples should not by coincidence include features that might mislead the learner. Finally, the teacher should introduce one essentially new feature per lesson and not try to teach multiple differences at once — a similar condition to Winston's "near miss" approach.

Intermediate results, and other implicit parts of examples, are difficult for a system to learn since they necessarily involve a large search. The system must consider everything that *might* be absent, and the numerous relationships it may have with other parts of the description. Andreae's (1984a, 1984b) "justified generalization" addresses this problem by using the amount of justification associated with a conjecture to determine which of three different levels of search to undertake. When the evidence is sufficiently strong, it will expend considerable resources seeking implicit relationships between parts of a description.

Although it does not increase formal learning power, the possibility of a system constructing its own examples and having them classified by an informant has considerable potential to speed up learning and reduce dependence on the skill of the teacher. This potential can only be realized if the system is able to synthesize "crucial examples" which discriminate effectively between alternative hypotheses consistent with the information seen so far. This ability seems to be one of the chief distinguishing characteristics of people who are good learners. To take the generation of examples one step further, if there is an automatic way of classifying examples as positive or negative, a system can attempt to learn autonomously through the paradigm of "learning by discovery".

CONCLUSION

Many present applications of concept learning techniques to knowledge acquisition for expert systems fall into the area of learning classification rules. For example, Carter & Catlett (1987) describe how a training set of 350 applications for credit cards, each classified as accepted or rejected, can be processed by ID3 into a decision tree which classifies around 80% of test cases "correctly" (ie according to the judgement of a human expert). Since the input is an attribute vector and the situation is noisy, top-down decision methods such as ID3 and PRISM are well suited to the task.

The version space technique represents a bottom-up approach. Its advantages are firstly in producing a result that is independent of the order of presentation of examples, and secondly in dealing with structured domains that cannot be captured by simple attribute-value vectors. These are offset by the fact that the version space collapses if the input is noisy. Moreover, the amount of computation *does* depend on the order of presentation of examples, and is very high in structured domains.

Hierarchical learning systems offer considerable promise but again suffer from excessive computational complexity and have only been applied to toy domains. A major research effort is required to integrate them with other techniques and use domain knowledge to constrain the search for alternative hypotheses which account for the examples seen so far.

Methods for function induction are ripe for exploitation in expert systems. Existing systems illustrate how constrained search can find the simplest input-output relation that accounts for given input-output tuples, and some (eg NODDY's) are not limited to purely numerical relationships. COPER shows how dimensional analysis can serve to determine the relevant arguments to functions, and how alternative bases can be searched to find economical expression of polynomial functions. Whether the same idea can be applied to more generalized typing and dimensional constraints is at present an open question.

Explanation-based generalization seems an attractive component for any knowledge-based system. Expert systems can be viewed as knowledge-intensive programs as opposed to domain-independent general-purpose problem solvers. Much work is required to build and debug the knowledge base of an expert system. This may take the form of extracting and operationalizing appropriate knowledge from a general theory. If so, an

alternative is to define the general knowledge needed for a problem-solver, but this is bound to be inefficient because it must search extensively for solutions. Explanation-based learning is able to operationalize just the knowledge needed to process typical examples (Pazzani, 1987b). This paradigm offers some promise for emulating the ability to learn from cases supplied by users, the hallmark of human expert behavior.

Finally, there are many opportunities for combining the various different learning methods. At the simplest, one can imagine a set of independent "learning experts", all receiving the same data, and reporting if and when they discover interesting structure. More interesting is the integration of different learning paradigms. Pazzani (1987a) describes an early experiment in which

- if an existing schema accounts for an event, it is indexed in memory by features which elaborate on that schema;
- otherwise, if existing schemata can be combined to explain it, a new schema is constructed using explanation-based learning;
- otherwise, if there are several similar examples, a new schema is created by similarity-based methods;
- otherwise, the event is simply added to memory.

Although this research is in a very early stage, it illustrates how different mechanisms can be brought into play depending on the knowledge available. Only in this way can we expect to build systems capable of sustained learning, continually expanding their knowledge base, drawing new lessons and insights from every example they encounter, the way real experts do.

ACKNOWLEDGEMENTS

We wish to thank Brian Gaines for suggesting that we write this paper and for constructive comments on an early draft. Financial support for this research has been provided by the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

Andreae, P.M. (1984a) "Constraint limited generalization: acquiring procedures from examples" *Proc American Association on Artificial Intelligence*, Austin, TX, August.

Andreae, P.M. (1984b) "Justified generalization: acquiring procedures from examples" PhD Thesis, Department of Electrical Engineering and Computer Science, MIT.

Angluin, D. and Smith, C.H. (1983) "Inductive inference: theory and methods" *Computing Surveys, 15* (3) 237-269, September.

Barr, A. and Feigenbaum, E.A. (Editors) (1982) *The handbook of artificial intelligence Volume II.* HeurisTech Press, Stanford, CA.

Boose, J. and Bradshaw, J.M. (1987) "Expertise transfer and complex problems" *Int J Man-Machine Studies, 26* (1) 3-28, January.

Bruner, J.S., Goodnow, J.J., and Austin, G.A. (1956) *A study of thinking.* Wiley, New York.

Carter, C. and Catlett, J. (1987) "Assessing credit card applications using machine learning" *IEEE Expert, 2* (3) 71-79, Fall.

Cendrowska, J. (in press) "PRISM: an algorithm for inducing modular rules" *Int J Man-Machine Studies.*

DeJong, G. and Mooney, R. (1986) "Explanation-based learning: an alternative view" *Machine Learning, 1* (2) 145-176.

Dennett, D.C. (1981) *Brainstorms.* Harvester Press, Brighton, Sussex.

Gold, E.M. (1967) "Language identification in the limit" *Information and Control, 10,* 447-474.

Haussler, D. (1987a) "Bias, version spaces, and Valiant's learning framework" *Proc 4th International Workshop on Machine Learning,* 324-336, Irvine, CA, June.

Haussler, D. (1987b) "Learning conjunctive concepts in structural domains" *Proc AAAI,* 466-470.

Hawkins, D. (1983) "An analysis of expert thinking" *Int J Man-Machine Studies, 18* (1) 1-47, January.

Hunt, E.B. (1962) *Concept learning: an information processing problem.* Wiley, New York.

Hunt, E.B., Marin, J., and Stone, P.J. (1966) *Experiments in induction.* Academic Press, New York.

Kokar, M.M. (1986) "Determining arguments of invariant functions" *Machine Learning, 1* (4) 403-422.

Krawchuk, B.J. (1987) "How to ask the right questions" Research Report, Department of Computer Science, University of Calgary.

Langley, P., Bradshaw, G.L., and Simon, H.A. (1983) "Rediscovering chemistry with the BACON system" in *Machine learning: an artificial intelligence approach,* edited by R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, pp 307-329. Tioga, Palo Alto, CA.

Langley, P., Zytkow, J.M., Simon, H.A., and Bradshaw, G.L. (1986) "Rediscovering chemistry with the BACON system" in *Machine learning Volume II,* edited by R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, pp 425-469. Morgan Kaufmann, Los Altos, CA.

Lenat, D.B. (1978) "The ubiquity of discovery" *Artificial Intelligence, 9,* 257-285.

Lenat, D.B. (1983) "EURISKO: a program that learns new heuristics and domain concepts" *Artificial Intelligence, 21,* 61-98.

Lenat, D.B. and Brown, J.S. (1984) "Why AM and EURISKO appear to work" *Artificial Intelligence, 23,* 269-294.

Michalski, R.S. (1983) "A theory and methodology of inductive learning" *Artificial Intelligence, 20* (3).

Mitchell, T.M. (1982) "Generalization as search" *Artificial Intelligence, 18,* 203-226.

Mitchell, T.M., Keller, R.M., and Kedar-Cabelli, S.T. (1986) "Explanation-based generalization: a unifying view" *Machine Learning, 1* (1) 47-80.

Niblett, T. and Bratko, I. (1986) "Learning decision rules in noisy domains" in *Research and development in expert systems III,* edited by M.A.Bramer, pp 25-34. Cambridge University Press, Cambridge, England.

Pazzani, M.J. (1987a) "Inducing causal and social theories: a prerequisite for explanation-based learning" *Proc 4th International Workshop on Machine Learning,* 230-241, Irvine, CA, June.

Pazzani, M.J. (1987b) "Failure-driven learning of fault diagnosis heuristics" *IEEE Trans Systems, Man and Cybernetics, SMC-17* (3) 380-394, May/June.

Quinlan, J.R. (1986a) "Induction of decision trees" *Machine Learning, 1* (1) 81-106.

Quinlan, J.R. (1986b) "The effect of noise on concept learning" in *Machine learning Volume 2,* edited by R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, pp 149-166. Morgan Kaufmann Inc, Los Altos, CA.

Sammut, C. and Banerji, R. (1983) "Hierarchical memories: an aid to concept learning" *Proc International Machine Learning Workshop,* 74-80, Allerton House, Monticello, IL, June 22-24.

Sammut, C. and Banerji, R. (1986) "Learning concepts by asking questions" in *Machine learning Volume 2,* edited by R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, pp 167-191. Morgan Kaufmann Inc, Los Altos, CA.

Schank, R.C., Collins, G.C. &A Hunter, L.E. (1986) "Transcending inductive category formation in learning" *Behavioral and Brain Sciences, 9* (4) 639-651, December.

Shapiro, E.Y. (1983) *Algorithmic program debugging.* MIT Press, Cambridge, MA.

Stepp, R.E. (1987) "Machine learning from structured objects" *Proc 4th International Workshop on Machine Learning,* 353-363, Irvine, CA, June.

VanLehn, K. (1983) "Felicity conditions for human skill acquisition: validating an AI-based theory" Research Report CIS-21, Xerox PARC, Palo Alto, CA, November.

Winston, P.H. (1975) "Learning structural descriptions from examples" in *The psychology of computer vision,* edited by P.H.Winston. McGraw Hill, New York, NY.

Witten, I.H. and MacDonald, B.A. (1987) "Concept learning: a practical tool for knowledge acquisition" *Proc Avignon 87 -- expert systems and their application,* 1535-1555, Avignon, France, May.

Wittgenstein, L. (1958) *Philosophical investigations.* Blackwell, Oxford, UK.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

HITTING A MOVING TARGET
(The Training of Knowledge Engineers)

J. Brian Woodward
Department of Computer Science
University of Calgary
Calgary,Alberta
Canada T2N 1N4

Abstract

The Knowledge Engineer is considered the prime designer of Knowledge-based systems. The Knowledge Engineering role requires skills and knowledge in all knowledge base development procedures from first concept to the final, functional system. However, with the advent of new tools for developing, editing and refining knowledge-based systems, the Knowledge Engineering role is undergoing redefinition. Many of the new developmental tools seek to reduce the need for a Knowledge Engineer. This situation strongly affects any program for the training of Knowledge Engineers. Knowledge Engineers are being trained but very few systematic accounts describing this activity are reported. This paper outlines one training component for a group of Knowledge Engineers and provides the assumptions made about the Knowledge Engineering role, the model underlying the training activity, description of the training activity, a critique of the course and issues arising from the training activity. Organizations wishing to provide such training must carefully define the role of the Knowledge Engineer in light of new tool availability and use

Introduction

There has been a major shift in computer technology and applications from data-processing to knowledge-processing. Expert systems and video-disc-based systems for instruction and marketing are examples of new application. Knowledge-based computing systems are recognized as having a major commercial significance. Major employment opportunities exist for those with the skills to develop knowledge-based systems but, so far, few training or job-experience possibilities are available. The background and training requirements for those developing such systems are quite different from those required for conventional data-processing activity. The development of knowledge-based systems requires a unique combination of skills.

This unique combination of skills can only be found in specific programs partcularly designed for the purpose: they are not generally available. Knowledge-Based Computer Applications is a joint University of Calgary project between the Knowledge Sciences Institute and the Computer Sport Systems Group in association with the Institute for Computer Assisted Learning. The project is funded by the Canadian Job Strategies Unit of the Department of Employment and Immigration. The purpose of the one-year project is to develop knowledge-based computer application specialists able to produce knowledge-based computer systems for industrial, commercial and educational applications. The overall training program is divided into 1. a formal component (courses, seminars, conferences and workshops) and 2. an on the job component (working groups with team leaders and project management sessions). The training component consisted of:

-elementary programming,
-introduction to hardware and communications,
-advanced programming,
-knowledge engineering,
-instructional design,
-computer applications in human performance, and
-introduction to interactive technology (videodisc).

This paper outlines the Knowledge Engineering course in detail and discusses the issues involved in training individuals in this area.

Knowledge Engineering

During a recent CAI conference, the administrative manager of the project was asked by an electrical engineer to define Knowledge Engineering. After receiving a description of the training activities, the engineer remarked sourly "That doesn't sound like engineering to me !". This statement and others like it may suggest that either Knowledge Engineering is not really 'engineering' in the usual sense of the word or that the activity described as Knowledge Engineering cannot be meaningfully communicated. It may also suggest that Knowledge Engineering, as a human activity system, is not clearly defined, or that the activity defies solid definition because it is still too new and developing. Whatever the case, Knowledge Engineering, both as an activity and a role may not be as clearly defined and understood as those doing the activity may wish.

A review of the definitions given to Knowledge Engineering by various authors in the field sheds some light on the situation. Initially coined by Feigenbaum, the term Knowledge Engineers refers to those individuals who build knowledge-based systems and refers to their activity and procedures as Knowledge Engineering. This general definition provides insufficient information to understand the function of Knowledge Engineering and the role of the Knowledge Engineer. More specifically, Bratko (1986) states that:

> "extracting some understanding of [a] domain from experts and literature and moulding this understanding into a chosen knowledge representation formalism is called the art of Knowledge Engineering." (p. 317)

These ideas are also succinctly put by Boose (1986), who states that Knowledge Engineering "is the term given to the extraction, articulation and computerization of expert knowledge". (p.11)

Harmon and King (1985), like most others, see Knowledge Engineering as an operational extension of AI. They state that given a variety of AI algorithms which view problems as a series of states and operators, and given methods of searching spaces, then Knowledge Engineers:

> "identify knowledge that can be applied to reduce the search space; develop knowledge frameworks as a mix of inference and representation; [and] implement [knowledge] as a system using appropriate language/tools. (p.6)

With reference to domain experts, these authors view Knowledge Engineers as:

> "specialists in getting the information from experts, prototyping an expert system and then working with the expert to improve the system."(p.6)

Can the activity described by these authors be called "engineering" ? This is an important question in the light of definitions given by other authors. Nagy, Gault and Nagy (1985) define a Knowledge Engineer as "a computer scientist who designs and builds expert system generators" (p. 288) and define a "knowledge-base author [as] the person who develops the code for the expert system based on the knowledge available" (p. 288). Rappaport (1986) uses the term "knowledge designer" for the individual who builds knowledge-based systems. Cairns & Woodward (in press) use the term "knowledge modeller" to describe someone who designs systems which are knowledge-based in nature. These descriptions blur the traditional definition but attempt to more accurately describe the activity called knowledge "engineering".

Also, the melding of two roles is becoming apparent: those of the Knowledge Engineer and the domain expert. This trend is supported by the development of a vast array of knowledge support systems:for example KITTEN (Shaw,1986), KSS0 (Gaines, 1987), AQUINAS (Boose & Bradshaw, 1987), and NEXPERT (Neuron Data, 1985) which are specifically designed to automate the "extraction, articulation and computerization" of knowledge from experts. Many specialized knowledge-based systems build the "extraction" process into the knowledge-based systems; as for example MORE (Eshelman et al, 1987) or SALT (Marcus, 1986).

Not only is the role of the Knowledge Engineer undergoing a re-definition, but those individuals who act in the role of Knowledge Engineer are demonstrating a wider variety of attributes. Initially, Knowledge Engineers developed from the ranks of computer scientists who had special interests in AI (Waterman, 1986; Bari and Feigenbaum, 1982). These individuals brought with them an AI focus to the field of Knowledge Engineering and this focus has dictated the development of the field. However, with the advent of new developmental tools, AI's hold on the Knowledge Engineering field is weakening. A wider variety of individuals are coming to the field, bringing with them foundations in other disciplines. In an introductory course on expert systems, the instructors found that the "best" expert systems were written not by the "brilliant foreign students or by engineering students" but by "psychology, communications and business majors" (Bahill and Ferrell, 1986).

D'Ambrosio (1985) suggests that a strict computer science background would be far too limiting for the Knowledge Engineer. Rather, what is needed is

> "a good, solid exposure to computer programming and computer science concepts together with a general liberal education."

The traditional AI-based definition of Knowledge Engineering appears in transition. Consequently the role of the Knowledge Engineer is undergoing redefinition and specific demands are being placed on those individuals assuming the role. The harder, quantitative side of Knowledge Engineering is giving way to the qualitative side, that of designing and modelling.

Training Knowledge Engineers

Programs designed to train individuals for Knowledge Engineering activities struggle to reflect the changing role demanded by new developments and appear slow to incorporate a broader interdisciplinary perspective. Given the AI focus of Knowledge Engineering training courses have strongly emphasized the "computerization" aspect."Articulation" has been severely restricted to the AI discipline and "extraction" of Knowledge has been largely ignored.

University-based training is usually in the form of Expert Systems courses (e.g. Bahill and Ferrell, 1986; Witten, 1987). These courses usually run for one semester and are taught as part of an engineering curriculum. Their focus tends to be on:

1. the design and construction of inference engines;
2. understanding heuristic programming methods;
3. AI-based concepts and research;
4. specific domain applications; and
5. programming in high level languages. (Bahill and Ferrell, 1986)

The DEC Knowledge Engineer course (La France, 1987) represents organization-based courses. This particular course is 14 weeks duration and focuses largely on the learning of a high level AI language and other available AI tools.

Expert System courses focus almost entirely on the "computerization" component with some emphasis on the "articulation" of very small problems. Although the Knowledge Engineering role demands "extraction" activity, little to no time is spent on developing experience or skill in this area. Also, other topics such as HCI or Instructional Design are seldom discussed or seen as relevant. The focus is entirely on representing knowledge in the machine and on the inference algorithms required.

A more recent development in training reflects more attention paid to the new technical developments. Bahill and Ferrell (1986) recognized that new expert system "shells" could be used as a basis for training. Their objectives were to:

1. understand the nature, limitations and suitable applications of expert systems;
2. effectively use expert system shells..; and
3. to produce small expert systems. (p. 59)

These authors also discuss the need for skill development in the knowledge "acquisition" end of knowledge-based systems development. They note that interpersonal skill is a strong asset.

This "tool-based" approach reflects the training activity of the vendors of various expert system shells. They offer short, very specific courses, on how to use their products for producing knowledge-based systems. D'Ambroisio (1985), for example, critiques a week long course on the M.1 product where activity centers largely on selecting problems which best suit the tool and on articulating how the tool works to solve the type of problem. Information about AI concepts and approaches are lightly discussed with some mention made of "acquisition".

It is apparent from the accounts of training activity that no course of instruction fully accommodates the demands of the Knowledge Engineer role. Books about knowledge-based systems offer another source of information which indirectly suggests the important areas of training for Knowledge Engineers. These sources, however, tend to follow the same strategy as training programs. There are many books on knowledge-based systems which focus moderately on the general articulation and computerization aspect of building knowledge-based systems (e.g. Klahr & Waterman, 1986; Waterman, 1986; Harmon & King, 1985; Hayes-Roth, Lenat & Waterman, 1983) or on the specific articulation and computerization aspect using one specific tool (e.g. Siegel, 1986; Nagy, Gault & Nagy, 1986; Weiss & Kulikowski, 1984). A notable exception to this pattern is the work by John Boose (1986) which focuses largely on the "extraction" activity using one specific tool (ETS).

Training Model

The purpose of this section is to articulate an approach to training Knowledge Engineers which attends to all phases of knowledge-based system development in line with the demands of the role as it changes. Although current Knowledge Engineering activity is highly AI-based, a broader perspective is becoming necessary and training models must reflect this perspective.

The Knowledge Engineering component of the project was based on the belief that Knowledge Engineering is essentially design and modelling activity. It is based on the assumption that the expert or the decision-maker has developed representations of the problem domain and that these models can, to some extent, be represented in software. The process is one of designing models of problems, solutions, decisions, and processes within the domain. The result is a software system which acts like the decision-maker or expert (Johnson, 1986).

Figure 1. outlines the model used to structure the training content and the guide the training activity. In order to model any knowledge-based activity, the domain requires specification. This domain space can be called a problem-space, decision-space, solution-space, decision-making space, or search-space as long as it clearly represents what activity is considered important within the domain. Within the domain are a series of objectives; clearly defined goal states which represent the result of a series of procedures and represent the purpose(s) of the knowledge-based activity within the domain. These objectives may be strongly or loosely linked with each other and either simultaneous or sequential in order. Within each objective is a series of tasks which represent discreet units of activity required to meet that objective. These tasks are ordered in a manner which best represents a decision-maker's approach to meeting that objective.

In order to represent the knowledge-based activity within each task, certain features must be elucidated and represented: 1. the type(s) of information (seen as knowledge-components) required and the role(s) this information plays in completing the task are essential components, as is 2. the source of this information and 3. the method(s) for eliciting it from experts/decision-makers. These three components of the knowledge-model represent the "human" side of the modelling activity in that the Knowledge Engineer must identify each feature and select the appropriate elicitation tools.

The "machine" modelling activity refers to the representation of the results of the "human" modelling in software form. The most appropriate knowledge-representation method for the information type is required. The role of the information is translated into inference strategies and control mechanisms. Finally, criteria are established for the knowledge base and knowledge processing activities for that task. These criteria include I/O screen designs and various performance criteria for task completion.

This model served two purposes in the training activity. It provided a conceptual framework for organizing course work and an operational model for building the knowledge-based systems required by the course.

Training Objectives and Activities

The following learning objectives for the course were established. The participants will:

KNOWLEDGE MODEL

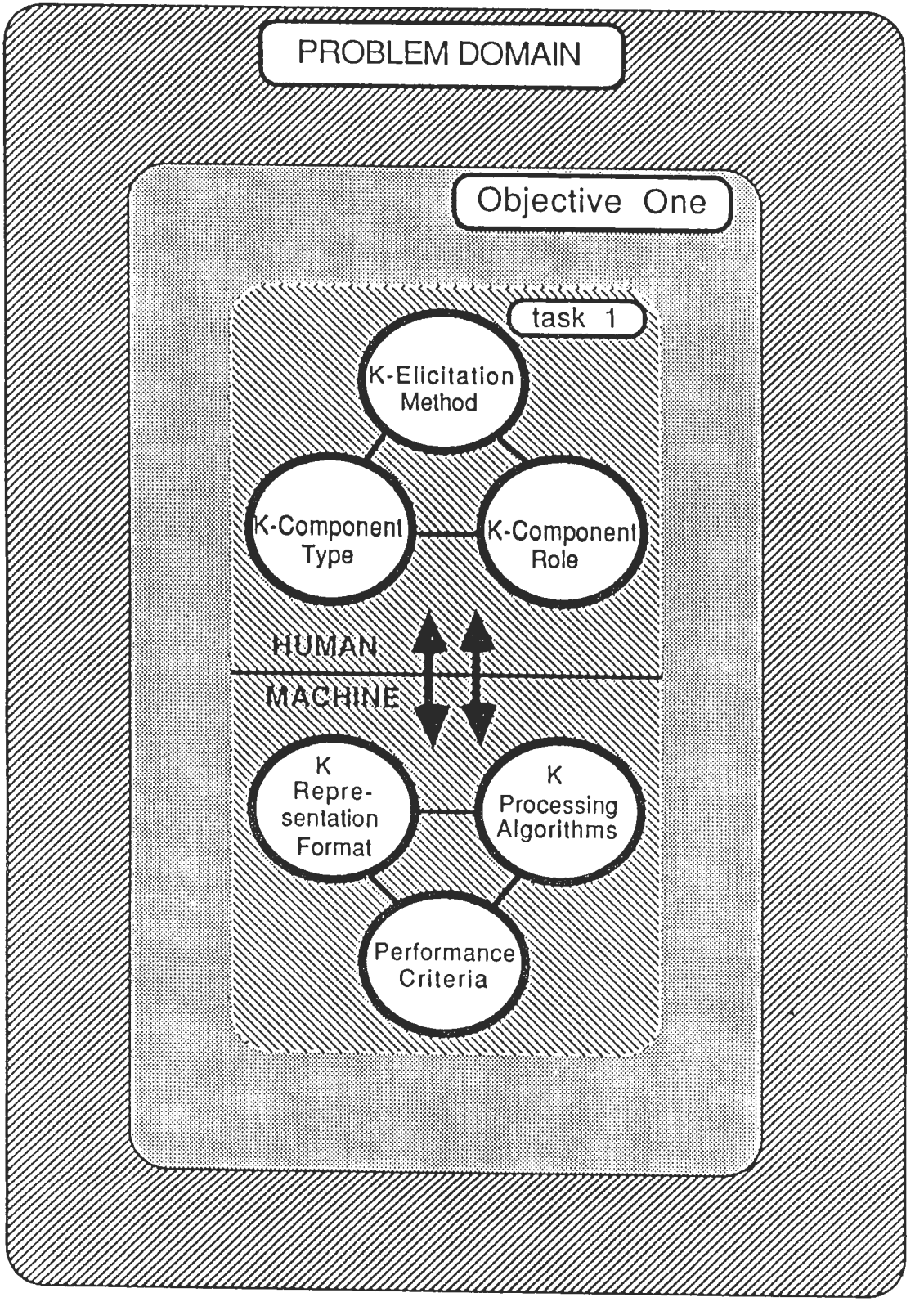


Figure 1.

1. relate the development of knowledge-based systems to the history and development of artificial intelligence;
2. be aware of the nature of human expertise and problem-solving;
3. be aware of existing expert systems, their structures and usage;
4. understand the nature, limitations and suitable application of expert systems;
5. identify and use appropriately different methods of knowledge representation;
6. identify and use appropriately different methods of logic and reasoning;
7. be knowledgeable about personal construct psychology and its relevance to the way in which people categorize experiences and classify their environment;
8. use repertory grid techniques to elicit and transfer expertise;
9. be aware of and use other techniques and tools for eliciting expertise;
10. effectively use one or more expert system shells; and
11. build a small expert system and debug it.

The course was scheduled for completion in four weeks. Due to project demands and the need for other course work, the four weeks were divided into three separate time intervals of one week, one week and two weeks. The weeks were not sequential.

Week One

The general objectives of this week were to introduce the participants to the area of Knowledge Engineering and to elaborate the concepts, methods, and procedures for the "human" modelling portion of knowledge-based system development. Mornings were given to lecture and discussion with afternoons being used for completing structured exercises arising from morning topics. Lectures covered the following topics:

1. history of knowledge-based systems and Knowledge Sciences as an emerging discipline;
2. expert systems and decision-support systems
3. the role of Knowledge Engineers
4. human expertise and problem-solving
5. the knowledge-modelling process: the mental models paradigm with personal construct psychology as an example
6. taxonomy of problem-solving: recognizing task demands
7. knowledge-elicitation methods
8. developing partial knowledge models

First week exercises required the participants to:

1. take a variety of problem statements and complete a task analysis according to the problem taxonomy of James Greeno (1984) and Hayes et al (1977). The objective was to familiarize the participants with recognizing various task demands embedded in problem solving situations in order to recognize more clearly what types of information are required and how those types are used in meeting the task demands.

2. split into 3 sub-groups of 3 members each to play the following roles: expert, knowledge engineer and observer. Using LaFrance's Knowledge Acquisition Grid 1986), the Knowledge Engineer completed two separate interviews with the expert and a task analysis of the problem. The observers recorded the difficulties apparent

in the activity. The reactions of each participant were reported and discussed in a debriefing session.

3. produce a partial knowledge-model based on task analysis and elicitation methods used by KITTEN (Shaw,1986) and KSS0 (Gaines,1987). The expert role was taken by two geophysicists and a media consultant who were participants in the program. The other participants assumed the Knowledge Engineering role.

Week Two

The overall objective for the second week was to elaborate on the concepts of the "machine" modelling activity. Morning lectures and afternoon exercise time continued to be the norm . Lectures covered the following topics:

1. structures of existing knowledge-based systems;
2. languages tools and development systems;
3. methods of knowledge representation;
4. methods of logic and reasoning;
5. prototyping knowledge-based systems;
6. performance criteria and debugging knowledge-based systems.

Second week exercises required the participants to:

1. produce a simple knowledge-based system similar to the Driver Responsibility problem outlined in Siegle (1986). Two person teams used a very simple expert system shell, MacKit (1985). This shell allowed only backward chaining inference, was rule-based, and was easy to learn. Each team was given the main components of the problem, thus reducing the need for knowledge acquisition. Each team was required to develop a rule base which would solve the problem. Aids for developing rules were used (Siegle, 1985; Nagy et al, 1986). Test cases and performance criteria were established early in the development. Each team's final system was presented and tested against six test cases and a set of ten performance criteria established by the instructor.

Weeks Three and Four

The objectives of these two weeks were first to introduce the use of a more sophisticated expert system shell and development tool, and second to build a more complex knowledge-based system based on a more complex problem. Lectures were kept to a minimum. The Nexpert (Neuron Data, 1985) system was presented and discussed and a review of the steps for building a knowledge-based system was undertaken. The remaining time was spent in representing a selected problem in knowledge-base form. The group was divided into teams to work on the following problems:

1. Velocity Analysis from Seismic Survey Data
2. Assessment of Professional Developmental Needs of Managers
3. Physical Fitness Advisor for Hockey Coaches

The final systems were presented and discussed under the following guidelines:

1. Scope of the Project
2. Performance Criteria for the Knowledge-base

3. Developmental Sequence
4. Algorithms Used
5. Problems Encountered/Solved
6. Performance Tests Used
7. Conclusions and Recommendations

Results and Critique

Instructor's Evaluation

There was a great deal of literature to review to meet the objectives of the course. The knowledge model presented earlier was of immense help in structuring the material for presentation. Extensive use of overheads and handouts helped in presenting the material. However, even though there were 120 hours total in the course and over 40 hours of lectures, time was insufficient to cover some topics to the depth required (e.g. handling uncertainty).

A frame-based shell was not available so activity was confined to rule-based representations. Participants would have benefited from exposure to such a system and the kinds of problems best suited to this form of representation.

The different shells, chosen for their difference in complexity, proved to be effective training tools for the most part. MacKit, as a raw shell, was very simple and an excellent first step, but its manual was almost useless. Nexpert proved to be an excellent developmental tool and shell, offering much flexibility and a well designed visual approach to activity. The only problems were that the manual was difficult to understand in many cases and the handling of uncertainty was cumbersome.

The building of two knowledge-based systems was demanding. Some participants stayed in the lab until late evening working on their systems. The final exercise required the selection of one or two objectives within the chosen domain rather than all objectives as first envisioned. While this change was somewhat disappointing to the participants, they learned to modularize or partition their problems cleanly, and to build such systems in stages.

All participants approached the tasks easily and quickly but those who had a basic grounding in logic seemed to fare better. Those participants with little computer experience or even a broadly based science background quickly hit a wall in understanding how inference procedures worked. Also, a programming background helped some participants develop algorithms for rule development and for logic strategies.

For the most part, those participants who could develop the logic of systems were not necessarily equally effective in the knowledge acquisition activity even when automated support tools were used. This result, suggesting a background in communication skill as essential, or at least extremely helpful for developing knowledge-based systems, is consistent with the observations of Bahill and Ferrell (1986). Also, participants with English as their second language had some difficulty in this area.

In contrast to the results obtained by Bahill and Ferrell (1986), course participants paid as much attention as possible to the user interface. This result was not unexpected since they had received a two week course in Instructional Design for CAI prior to building their first knowledge-based system.

Participant's Evaluation

The original course objectives were used as a focus for student evaluations. Reaction to the course was requested in two forms for each onjective: a score on a seven point rating scale from 1 (high) to 7 (low) and a written reaction in the form of critique and recommendation. For the most part, the course met the objectives to a fair to good extent (average 2.5 on the 7 point scale). Objectives 1 and 9, however, averaged 3.2 and 3.3 respectively suggesting that these to objectives were not met to the same extent as others.

Comments made by the participants were extremely useful. The following points summarize the various reactions to the course. Participants' comments suggested that:

1. a much longer course (up to four months) be offered rather than a four week course and suggested that the four weeks be kept together in sequence;

2. the use of a simple shell was a very useful stepping stone prior to using a more sophisticated tool;

3. there was not enough time (although lots of interest) to use all of the procedures available in Nexpert;

4. the first week resulted in "information overload" from all the new concepts and ideas;

5. maybe the knowledge acquisition procedures should be presented after knowledge-based systems were described in detail;

6. a model for deciding what problem would be suitable for representation in a knowledge-based system would be of benefit;

7. poor documentation for the shells added to frustration given the time limit of the course;

8. more time could be spent on how the information from automated knowledge elicitation methods is used in the development of knowledge-based systems;

9. case studies of small problems should be used as a basis for understanding how to match various modelling techniques with the problem demands;

10. more background in the Cognitive Science field would have been beneficial; and

11. the non-threatening aspect of the MacIntosh environment was very helpful;

Issues and Recommendations

What constitutes the domain of Knowledge Engineer instruction and development? The field of AI has acted as the birth environment for the field, but as Knowledge Engineering becomes hardened-off on real-world problems the domain of Knowledge Engineering will broaden. This paper outlines a more comprehensive approach with a broader curriculum than has been previously reported.

As with any field undergoing development, there is a dirth of essential teaching materials. For example, a book of case studies which outline various types of problem representations would be useful; however, the cases would need to be in every-day language as opposed to domain specific vocabulary. Materials which help guide the selection of an appropriate method of formal representation of a given problem would be

very useful. Work has emerged in this area (e.g. Bylander and Chandrasekaran, 1986; Bylander and Mittal, 1986; Bennett, 1983) but has not yet been consolidated in a form for training.

How long does it take to train a Knowledge Engineer? This question can be answered tritely but it reflects an important issue. It is not clear what would constitute an answer to this question. Accounts of "advanced" training in knowledge-base development are lacking as are descriptions of "introductory" courses.

What skills, abilities, and even attitudes are prerequisites for the role of Knowledge Engineer? What tools, skills, and abilities are necessary and sufficient for the Knowledge Engineering role ? As more developmental tools become available, the Knowledge Engineer role changes. Basic skills require elucidation.

What constitutes evaluation in and of this training activity ?
Are university models which focus on grading and evaluation best? Are organizational training models which focus heavily on the transfer and cost effectiveness of training useful?

What best describes the role of the Knowledge Engineer? It might best be re-construed as Knowledge Design and Knowledge Modelling, broadening the definition of the role and removing it from the sole confines of AI. In doing so, the role may take into account such diverse activities as videodisc design, integrated systems, knowledge-based training systems and hypermedia systems. This redefinition leads to a more stable role and consequently to a human activity system which can be better defined and elucidated. Training guidelines, basic skills, curricula, and evaluation criteria could be better organized and operationally defined.

Bibliography

Bahill, A.T. and Ferrell, W.R. (1986) Teaching an introductory course in expert systems. **IEEE Expert, Vol. 1** (4) 59-63.

Bari, A. and Feigenbaum, E. (1982) **Handbook of Artificial Intelligence: Volume II.** William Kaufmann Inc.: Los Altos, California.

Bennett, J.S. (1983) ROGET: a knowledge-base consultant for acquiring the conceptual structure of an expert system. **Report HPP-83-24, Computer Science Department, Stanford** University: Stanford, California.

Bratko, I. (1986) **Prolog: Programming for Artificial Intelligence.** Addison-Wesley: Don Mills, Ontario, Canada.

Boose, J.H. and Bradshaw, J.M. (1987) Expertise transfer and complex problems: using AQUINAS as a knowledge-acquisition workbench for knowledge-based systems. **International Journal of Man-Machine Studies,Vol. 26** (1), 3-28.

Boose, J.H. (1986) **Elicitation Techniques for Expert System Design**. Elsevier: New York.

Bylander, T. and Chandrasekaran, B. (1987). Generic tasks for knowledge-based reasoning: the "right" level of abstraction for knowledge acquisition. **International Journal of Man-Machine Studies,Vol. 26** (2), 231-244.

Bylander, T and Mittal, S (1986). CSRL: a language for classificatory problem solving and ncertainty handling. **AI Magazine, Vol. 7** (2), 66-77.

Cairns, K.V. and Woodward, J.B. (in press) Life choices simulation: model and methodology. **Systems Practice, 1** (1).

D'Ambrosio, B. (1985) Building expert systems with M.1. **BYTE, Vol.** 10 (6).

Eshelman, L., Ehret, D., McDermott, J. and Tan, M. (1987) MOLE: a tenacious knowledge acquisition tool. **International Journal of Man-Machine Studies,Vol. 26** (1), 41-54.

Gaines, B.R. (1987) Rapid prototyping for expert systems. Oliff, M. (ed.) **Proceedings of International Conference on Expert Systems and the Leading Edge in Production Planning and Control.** pp.213-241. University of South Carolina.

Greeno, J. G., Riley, M.S. and Gelman, R.(1984) Conceptual competence **Cognitive Psychology, 16,** 94-143.

Harmon, P. and King, D. (1985) **Expert Systems: Artificial Intelligence in Business**. John Wiley and Sons, Inc.: Toronto.

Hayes-Roth, F., Lenat, D.B. and Waterman, D.A. (eds.) (1983) **Building Expert Systems.** Addison-Wesley: Reading, MA.

Hayes, J.R, Waterman, D.A. and Robinson, C.S. (1977) Identifying the relevant aspects of a problem text. **Cognitive Science, 1,** 297-313.

Johnson, P.E. (1986) Cognitive models of expertise. Paper presented at **Symposium on Expert Systems and Auditor Judgment.** University of Southern California.

Klahr, P. and Waterman, D. (1986) **Expert Systems: Techniques, Tools and Applications.** Rand Corporation: Reading, MA.

LaFrance, M. (1987) The knowledge acquisition grid: a method for training knowledge engineers. **International Journal of Man-Machine Studies,Vol. 26** (2), 245-256.

Mackit Manual: **Introduction to Knowledge Engineering** (1985). Knowledge Systems Environments Inc., Dillsburg, PA.

Marcus, S. (1986) Taking Backtracking with a grain of SALT. **Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop:** Banff, Canada.

Nagy, T., Gault, D. and Nagy, M. (1985) **Building Your First Expert System.** Ashton-Tate Publishing: Culver City, California.

Nexpert Manual: **Nexpert Fundamentals** (1985). Neuron Data, Palo Alto, California.

Rappaport, A. (1986) Multiple Problem Spaces in the Knowledge Design Process. **Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop:** Banff, Canada.

Shaw, M.L.G, and Gaines, B.R. (1986) Techniques for knowledge acquisition and transfer. **Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop**: Banff, Canada.

Siegel, P. (1986) **Expert Systems: A Non-programmers Guide to Development and Applications**. TAB, Blue-Ridge Summit, PA.

Waterman, D.A. (1986) **A Guide to Expert Systems.** Addison-Wesley: Don Mills, Ontario.

Weiss, S. and Kulikowski, C. (1984) **A Practical Guide to Designing Expert Systems.** Rowman and Allanheld: Totowa, NJ.

Witten, I. (1987) A course on "Expert Systems" for electrical engineering students. **Proceedings of the 18th SIGCSE Technical Symposium on Computer Science Education.** St. Louis, Missouri.

2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,
Banff, Canada, Oct., 1987

Design Goals for Sloppy Modeling Systems

Stefan Wrobel

Techn. Univ. Berlin, FR 5-8
Franklinstr. 28/29
1000 Berlin 10
West Germany

E-mail: wrobel%db0tui11.bitnet@wiscvm.wisc.edu

Abstract

The *sloppy modeling* paradigm regards knowledge acquisition as a cooperative process between user and system, in which the system's learning and structuring algorithms and the user are regarded as partners in a common problem solving activity. In this paper, we discuss the design goals that this paradigm entails for a knowledge acquisition system, with a special focus on the environment that needs to be presented to a user. We identify six criteria that such a sloppy modeling environment should meet. We then present the sloppy modeling system BLIP, its knowledge representation, its system architecture, and its user interface as an example of such an environment. After a transcript of a sample session with the system, we finally evaluate BLIP on the criteria established.

1. Introduction

Knowledge acquisition has long been recognized as one of the most difficult parts of building a knowledge-based system. The knowledge that is available from an expert or other source must be represented and structured in such a way that it becomes usable for the system. Several approaches exist for supporting the knowledge acquisition process, including, to name only a few examples, systems that are oriented towards knowledge editing, such as MOLE [Eshelman/McDermott 86], knowledge elicitation systems such as AQUINAS [Boose/Bradshaw 86], and inductive (AQ, [Michalski/Chilauski 80]) and apprentice learning systems (LEAP, [Mitchell et al. 85]).

In the KIT-LERNER project[1], we are developing a knowledge acquisition paradigm that we have termed *sloppy modeling* (or *cooperative modeling*). In the sloppy modeling paradigm, knowledge acquisition is viewed as a cooperative modeling process between user and system. This means that the user is not required to develop a complete and well-structured model beforehand in order to then transfer it into the machine. Instead, the

[1] The KIT-LERNER project is partially supported by the German Ministry of Research and Technology (BFMT) under contract ITW8501B1. Industrial partners are Nixdorf Computer AG and Stollmann GmbH.

modeling activity itself becomes part of the system-supported knowledge acquisition process: selecting the proper abstraction/granularity level for the knowledge to be represented, choosing the right concepts, determining their interactions, etc. In that respect, sloppy modeling shares a common view with advanced knowledge elicitation systems (e.g., AQUINAS), and machine learning approaches, where the above tasks are system-supported as well.

It differs from those approaches in its emphasis on a cooperative, mixed-initiative modeling process. In a knowledge elicitation system, the user is guided through a more or less fixed dialogue, answering the questions posed by the system. There is no way for the user to take the initiative, to go back and reverse decisions made earlier in the process without basically starting anew from scratch. In the sloppy modeling paradigm, the goal is a modeling style where the user and the system's structuring and learning routines are regarded as independent partners that both work towards having a well-structured and complete model.

A sloppy modeling system offers knowledge editing facilities that can be used freely and whenever the user wants. The system's part in the modeling process is to try to organize and complete the knowledge entered by the user. To that end, such a system uses knowledge analysis and learning techniques to discover structures and concepts that the user may not have explicited, and introduces them into the emerging model using the same constructs that are available to the user. The user, in turn, is provided feedback on the actions of the system, thus influencing his or her own modeling activities.

Our ideas about sloppy modeling are embodied in a computer program called BLIP. As a knowledge acquisition system, BLIP is designed to acquire *encyclopedical knowledge*. Encyclopedical knowledge is the basic problem-solving independent knowledge about a domain including terminological and simple empirical knowledge. In order to use such knowledge in a performance system, problem-solving and consultation knowledge has to be added or compiled in, as is proposed within the Explainable Expert System (EES) paradigm [Neches et al. 85].[2] Alternatively, BLIP can be used directly with performance systems that support an explicit separation of knowledge into encyclopedical, problem-solving, and consultation knowledge (cf. HERACLES, [Clancey 86]). BLIP offers comprehensive facilities for entering, inspecting, and manipulating encyclopedical knowledge. Sloppy modeling is supported by using machine learning techniques that discover inferential relations between existing knowledge [Morik 86a].

A major focus of work on BLIP has been the development of an environment that will permit a sloppy modeling type of interaction with the system. In this paper, we will first discuss some of the design goals brought about by assuming a sloppy-modeling paradigm. In section 3, we will then present BLIP's knowledge acquisition environment and the modeling operations it supports. Those operations will then be illustrated by a sample session with the system in section 4. In section 5, we will then discuss BLIP's strengths and weaknesses with respect to the sloppy modeling design goals put forth in section 2. The learning techniques that constitute the system's part of the sloppy modeling process are not subject of this paper.

---

2 For more detail on the motivations for choosing such an approach, we refer the interested reader to [Emde/Morik 86].

2. Design goals for a sloppy modeling environment

The central idea of sloppy modeling is to look at knowledge acquisition as a model building process that is to be supported by the acquisition system. In other words, the "paper-and-pencil" phase of structuring and organizing a domain is to be performed directly at the computer terminal with the help of the sloppy modeling system. We have identified several requirements that a system must meet to allow such a style of interaction (Figure 1).

- Flexibility
- Reversability
- Integrity Maintenance
- Liveness
- Inspectability
- Transitionality

Figure 1: Requirements for sloppy modeling environments

Flexibility. In the first phase of modeling a domain, the human modeler (knowledge engineer, expert) has not yet built up the structures and concepts that would allow him or her to enter his or her knowledge in an organized fashion into the system. Rather, that first phase will be characterized by many sudden ideas of what ought to go into the model and how it should be structured. Correspondingly, the user's focus of attention will change quite often, suspending an ongoing activity to do something else first, and then returning to the original activity. To allow a user such an unconstrained interaction, the modeling system must be able to use information whenever it becomes available instead of disturbing the user when it is needed. It should not involve the user in extended structured dialogues, but instead offer small operations that can be done at any time.

Reversability. Like flexibility, the reversability requirement is a consequence of the nature of the first modeling phase. Since at any point, it may become necessary to change content or structure of the model, the knowledge acquisition system must support not only monotonic incrementality, i.e., additions to existing knowledge, but also changes and deletions. Those modifications should be supported by appropriate operations to facilitate substantial changes to a model. Besides requiring reason maintenance operations on the knowledge base itself, the reversability criterion poses difficult requirements for the learning components of a sloppy modeling system, because inductive or default assumptions might need to be retracted as well, requiring sophisticated knowledge revision procedures.

Integrity and Consistency Maintenance. In a knowledge acquisition system with several different knowledge sources that depend on each other, a difficult problem is to assure that entries in one knowledge source are consistent with entries in other knowledge sources, and propagate changes accordingly. To distinguish this problem from the (separate) problem of logical consistency, i.e., avoiding logical contradictions in the model, we use the term *integrity* instead. In many knowledge acquisition systems, integrity maintenance is largely left to the responsibility of the user. This may be acceptable in situations where knowledge acquisition is seen as the "coding" of an existing model. In a sloppy modeling environment, it constitutes a burden that detracts

the user from the real task of coming up with a model: integrity maintenance is a system-level, not a knowledge-level ([Newell 82]) activity. Therefore, a sloppy modeling system should support the user by monitoring integrity constraints, resolving some conflicts automatically, and allow the user to handle the others at a convenient moment. Of course, similar support should be available for consistency maintenance.

Liveness. One problem people have in modeling a domain is to keep track of the consequences of their modeling decisions. The introduction or deletion of one piece of knowledge may have far-reaching consequences for the rest of the model. In sloppy modeling, this problem is further exacerbated, because not only does the user change the model, but the system does so as well. It is therefore crucial that a sloppy modeling system offer a *live* user interface, i.e., an interface that provides immediate feedback on the consequences of an operation with respect to all parts of the modeling system. Instead of requiring an edit-process-debug cycle, the inferencing and learning components of the modeling system should immediately react to changes to the model. For the graphical displays of a model's structure, liveness is a very important property as well.

Inspectability. Requiring transparency and inspectability for a knowledge acquisition or representation system is universally accepted by now, and browsing and other graphical techniques are standard features of many systems. During the first phase of modeling, such aids are even more important for developing a "feel" for a domain and how to model it. A second aspect of inspectability is support for working with larger domain models, where the entire model cannot be in view at once. There, we need a conceptually oriented way of focusing what is displayed: depending on the activity being performed, the relevant pieces of knowledge should automatically be displayed. Relevancy, of course, must be defined in terms of the interconnections within and between knowledge sources.

Transitionality. As a final criterion, we require a sloppy modeling system to be *transitional* in two senses. First, as a general goal for all computer systems, the system should be structured in such a way that a novice user can get to know the advanced interaction modes with the system by practicing with the simple ones [Badre 84]. For a knowledge acquisition system, this means that a user must be able to start working in a domain-oriented style (at the "knowledge level", [Newell 82]), and gradually get to know the more advanced system-level facilities.
Second, and more importantly, a sloppy modeling system must offer a continuous transition from a purely user-controlled modeling style, where the system does not make any changes to the model, to a mixed-initiative modeling style, where the learning components of the system assist the user in structuring and extending the model. This requires using the same interface, knowledge sources and representations for both user-controlled and mixed-initiative modeling.

3. Sloppy modeling with BLIP

The above design goals are independent of the particular knowledge representation and modeling support strategies chosen for a sloppy modeling system. The knowledge acquisition interface for BLIP that we are about to present is an example of how those requirements can be realized in a knowledge acquisition system based on a logic-oriented representation.

System overview

BLIP's knowledge representation is based on a sorted logic augmented with selected higher-order concepts. The basic domain-level entities that a user works with are *facts*, *predicates*, and *rules*. A fact is simply a predicate with arguments, such as

contains(aspirin,asa),

which could be paraphrased as "Aspirin contains Acetyl Salicyl Acid."[3] Each predicate has a definition specifying (among other things) its *argument sort mask* (ASM), which is used to assign terms (naming domain objects) to sorts. The predicate "contains", which was used above, might have the ASM

contains(<drug>,<substance>).

Integrity constraints assure that each predicate used in a fact has a proper definition and each term is properly assigned to a sort in the sort hierarchy. No term can be a member of more than one sort (except super/subsorts).

Inference rules in BLIP are expressed by using the system's *metapredicates*. A metapredicate expresses a property of a (domain level) predicate or a relation between (domain level) predicates, which is defined by the metapredicate's associated *rule schema*. The metapredicate "inverse", for example, expresses the relation of two predicates being the inverse of each other:

inverse(p,q): p(x,y) --> q(y,x).

where p and q are predicate variables. Using this metapredicate definition, a *metafact*, such as

inverse(contains,is_contained_in),

can be translated into the domain level rule

x,y :: contains(x,y) --> is_contained_in(y,x).

By expressing all rules as metafacts, inferences can be made about rules by using metarules/metametafacts. That also provides a solution to the problem of maintaining consistency among a set of rules, which is difficult otherwise. Again, it is an integrity constraint that each metapredicate used in a metafact have a proper definition.

Just as metapredicates describe properties of predicates, *metametapredicates* (actually, *metametafacts*), are used to describe properties of metapredicates. Metapredicates, metametapredicates and metametafacts constitute the domain-independent knowledge that BLIP is equipped with. It is used for consistency maintenance and for the learning process that tries to extend an emerging domain model by inducing rules (see [Morik86b] for more details about metaknowledge).

An important point is that, although the interface offers the necessary operations, a user never needs to explicitly manipulate metaknowledge. Metaknowledge can be considered a system-level knowledge representation construct that a domain-oriented user does not

---

[3] Examples are taken from a domain about the side effects of pain killers, which was modelled following [Langbein et al.].

need to be aware of. As we will see below, a user can simply work with facts, predicates and rules - the latter will automatically be reformulated as metafacts, extending the set of available metapredicates, if necessary.

Facts and rules at the domain, meta, and metameta levels are stored in BLIP's *inference engine* [Emde forthcoming], which also carries out the necessary reason maintenance operations. Sorts and predicate definitions are stored in separate knowledge sources and maintained by the system's *coordinator* module, whose main task is to maintain integrity between knowledge sources, and to manage the interactions between the different system parts, i.e., interface, inference engine, and modeler (Figure 2). The modeler contains the learning and model restructuring processes. (For more information on BLIP's knowledge sources and on its architecture, see [Emde/Morik 86]).

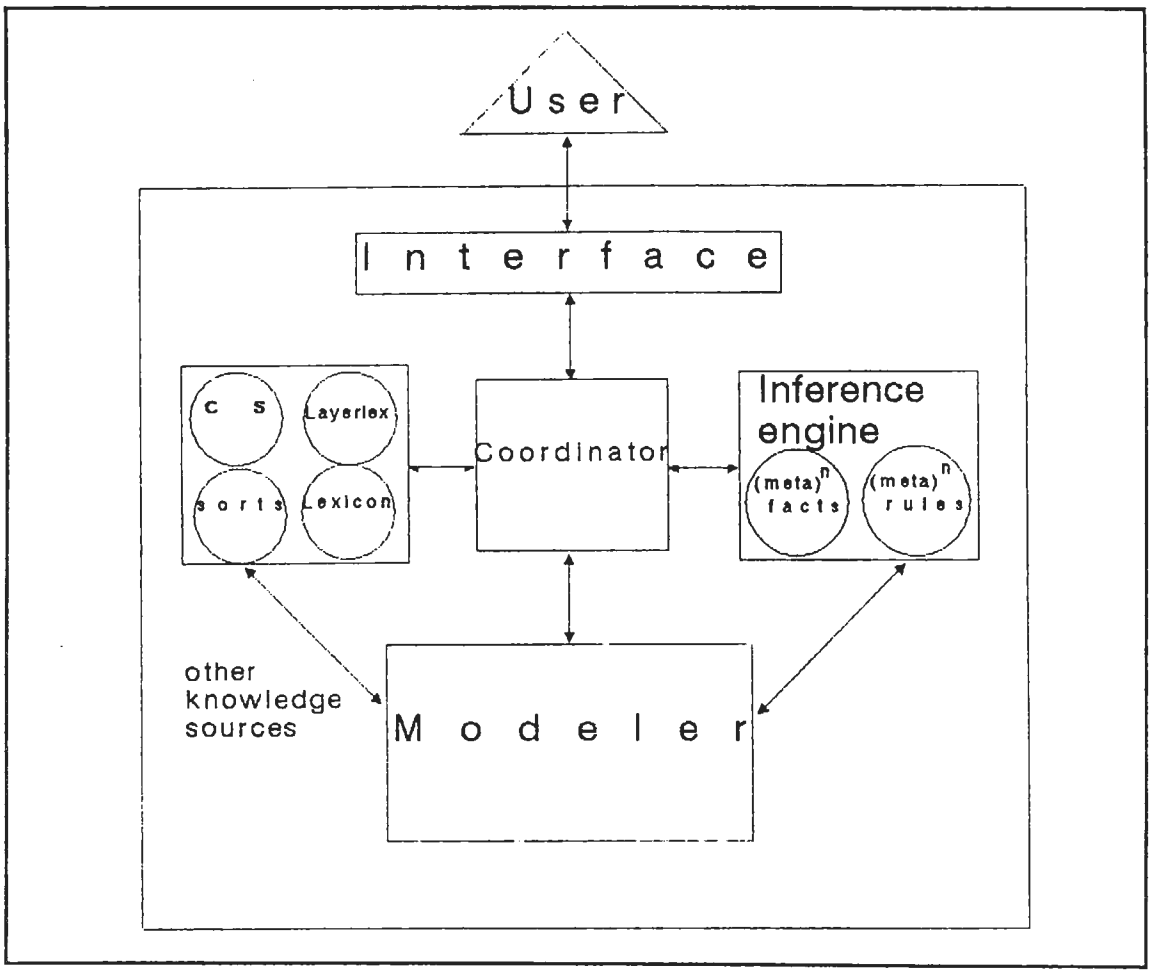


Figure 2: System Architecture

Basic interface concepts

BLIP's knowledge acquisition interface consists of a display with three *physical windows* which can be configured to show any one of the *logical windows* that correspond to the different knowledge sources of the system (Figure 3). The basic way of displaying a knowledge source is as text by using scrollable windows with mouse-sensitive items representing entries in the knowledge source. In Figure 3, the BLIP display screen is configured to show the *facts*, *predicate*, and *sorts* logical windows. For general (non-item specific) operations on a knowledge source, each of the display areas has its own command area offering commands as mouse-sensitive items. In addition to the three display areas, the BLIP screen also contains two "global" command areas which offer system-wide operations, such as saving or loading a domain.

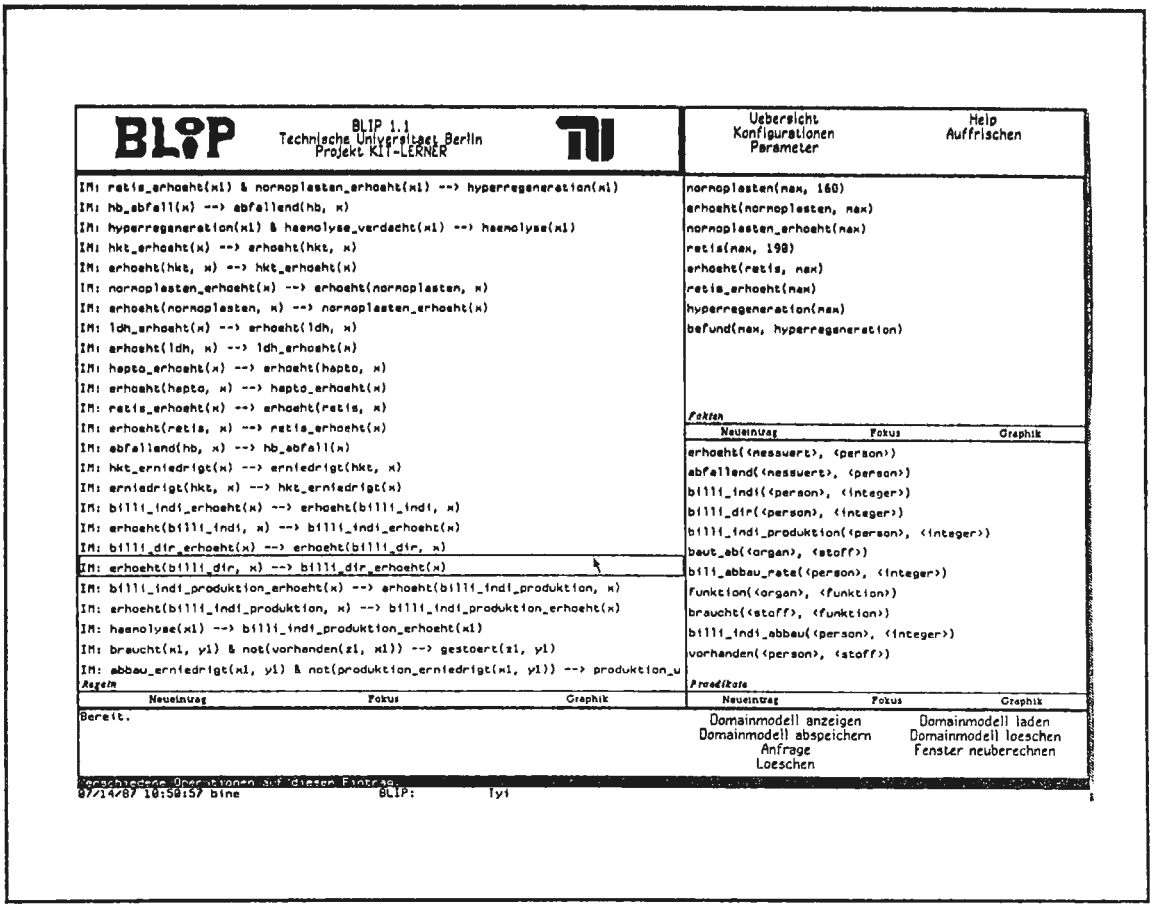


Figure 3: BLIP's display screen

As called for by the flexibility criterion for sloppy modeling systems, BLIP does not offer prestructured activity sequences. Instead, any of the operations are available to the user at any time. The user may scroll the display windows to inspect a knowledge source, or reconfigure the screen to show different logical windows. By clicking on a window item, a menu can be brought up that offers operations on that item. Alternatively, any of the knowledge source-related or system-wide commands may be selected.

Note that the interface is strictly organized around the knowledge sources that comprise a

domain model in BLIP. Since the knowledge sources used by the learning and restructuring algorithms are the same that are manipulated by the user, there is no need for windows providing access to those processes.

Basic knowledge editing and management facilities

The basic editing facilities that are offered include adding a new entry, copying and modifying an existing entry, and deleting an entry. Those operations may be selected from the menu that comes up when clicking on an item in a window, or (the new-entry command) from the command area at the bottom of the window that displays the knowledge source. The actual entry or modification is accomplished by filling in or modifying entries in a form oriented at the structure of the item to be edited. That way, syntactically malformed entries are excluded.

As soon as an editing operation is completed, the addition or change in the system's knowledge is processed by both inference engine and modeler, and changes caused by reason maintenance or learning procedures are immediately propagated into the appropriate interface windows. That way, the user gets an immediate feedback about the consequences of an editing operation. Contradictions between new inputs and existing knowledge are detected and the user notified as well.

Besides using the display windows for inspecting the system's knowledge, the user can also formulate direct queries to the inference engine, which may cause additional deductions.[4] The inference engine also offers a backtrace of applied inference rules as a primitive explanation facility. Due to the reason maintenance procedures, the editing operations are completely incremental and undoable (although not automatically at this point).

Integrity maintenance

In any knowledge representation system with several interdependent knowledge sources, integrity maintenance is a hard problem. In BLIP, we need to enforce the constraints that each predicate used have a definition (at the domain, meta, and metameta levels), that each term be a member of exactly one sort, etc.

Unfortunately, even simply enforcing those constraints is not an easy task if the user is to be offered the flexibility and reversability necessary for sloppy modeling systems. What is to be done, for example, if a user tries to delete a predicate while there are still some facts that use it? Integrity maintenance needs to make sure that the deletion is either not performed or all facts are deleted as well. In a sloppy modeling system, things are even more difficult, since immediately enforcing integrity constraints distracts the user from the task of modeling and is therefore undesirable.

The solution we have found in BLIP is to manage integrity maintenance via an *agenda* of "open ends", i.e., integrity violations that still need to be taken care of. When an integrity violation occurs, the user is notified and at the same time, an agenda entry is generated. The offending entry is processed as far as possible, and then stored on the agenda. The user may then choose to act immediately or wait till a later point. When the user has

---

[4] Forward chaining inferences from inputs have a depth limit, so that some inferrable facts will not be derived immediately, but only on query.

removed the integrity constraint, the agenda item is removed automatically and the remaining operations that had to be suspended earlier are carried out. If, for example, a user uses a predicate in a fact without defining it first, then that fact can be entered, but the sort assignments have to be delayed until the predicate definition is known.

Besides being difficult to handle when they are violated, integrity constraints can also be exploited advantageously in some situations. In BLIP, they are used (a) to automatically generate predicate definitions, and (b) to automatically generate a sort hierarchy.

Generation of predicate definitions. If a new fact with an undefined predicate is input, and the sorts of its argument terms are already known, we can exploit the constraint that each term belongs to one sort only and automatically generate the predicate's argument sort mask.

Generating a sort hierarchy. That same constraint can be used to come up with a sort hierarchy. If a term is used in two different sorts, the system checks whether one sort (the smaller one) can be made a subsort of another. Only if that is impossible an integrity constraint violation is signalled. If the extensions of the sorts later change, the system automatically restructures the sort hierarchy accordingly.

Knowledge Analysis Support

Graphics. The basic, text-based way of looking at a knowledge source is often quite insufficient for developing a feel for the domain. We therefore added graphical display routines similar to the browsers of several knowledge representation systems (e.g., BACK [Peltason et al 87]). Just as the text displays, the graphics are mouse-sensitive and can be used to select items and invoke commands like focusing, etc.

The Focus Facility. In larger domains, the size constraints of the display usually don't allow showing the entire contents of a knowledge source in a window, and scrolling through a window is a cumbersome process. Fortunately, one hardly ever needs to see the entire domain; usually, one concentrates on a certain part of the domain, so that it is sufficient to show all entries related to that part of the domain. In BLIP, we have implemented a simple focus concept, which allows the user to select the knowledge source entries to be shown based on their main functors and arguments.

Automatic generation of metafacts and metapredicates

An important goal in the development of BLIP was that the system be usable without intimate knowledge of the more system-internal knowledge representation constructs like metapredicates, just by working with facts, predicates, and inference rules. For facts and predicates, this poses no problem. Recall, however, that rules must be expressed by using the system's metapredicates. In order to insulate a domain-oriented user from those metapredicates, BLIP contains procedures that automatically reformulate an input rule as a metafact. To that end, the rule is compared to the ruleschemata of all known metapredicates to see if they can be used to express the rule. If an appropriate one is found, the corresponding metafact is entered into the inference engine. If more than one

metafact is found,[5] the system lets the user choose or picks a metafact at random. The latter possibility is not quite as bad as it seems, because if the chosen metapredicate produces erroneous conclusions, these will be discovered by the reason maintenance procedure anyway and can then be handled.

A more interesting problem occurs if no appropriate metapredicate is found. In that case, the system defines a new metapredicate by using the given rule as an example and abstracting from the concrete predicates used in the rule. The user is given an option to name the new metapredicate. An open problem is how metametafacts that describe the properties of new metapredicates can be generated (such metametafacts are used, among other things, for consistency maintenance). Currently, as a simple aid to the user, a list of all metafacts whose corresponding rules are syntactically compatible with the new metapredicate is generated and offered to the user for selection.

4. A Session with BLIP

In this section, we will take the reader through a modeling session with BLIP to demonstrate in context the system features that were described in the previous section. Our goal in this modeling session will be to represent some simple knowledge about conferences and workshops - a trivial domain, but sufficient to demonstrate many of BLIP's features. In the simulated BLIP display screens shown below, the most recent user input is marked with a backfacing arrow ("<="). All items that are new in a particular screen (the user input and its consequences) are underlined, and all items that were deleted from the previous screen are ~~struck out~~.'

Normally, when a user starts building up a domain model, BLIP begins with a significant amount of metaknowledge already built in. Here, we will start with an empty domain model instead, thus avoiding interactions with previously entered knowledge. This will also give us a chance to see how new metaknowledge can be created. We therefore start the session by clearing the knowledge base, and selecting a configuration of logical windows we want displayed on the screen; it shows the *Facts*, *Predicates* and *Sorts* logical windows. We're now ready to start entering knowledge into the system. We begin by defining a new predicate which expresses that an event is deemed "large" (Screen 1).

---

5 This can happen, because metapredicates are defined not only by their ruleschemata, but also by the metametaknowledge available about them: "inclusive" and "equivalent" both have the rule schema "p(x) --> q(x)", and the difference between them is that "equivalent" is symmetrical (which is expressed as a metametafact) and "inclusive" isn't.

BLIP TU	
	large(<event>) <= *Predicates*
	subsort(event,all) sort(event,[]) *Sorts*
Facts	

Screen 1

The new entry appears on the predicate window, and, more interestingly, we also observe a change in the sorts window that immediately reflects the introduction of the sort "event" in the newly defined predicate. "Event" is recorded as having no members and being a subsort of "all", the set of all terms. This already illustrates the concept of a *live* interface - any changes are immediately propagated to all knowledge sources by the inferencing component.

BLIP TU	
large(ijcai) <=	large(<event>) small(<event>) *Predicates*
	subsort(event,all) sort(event,[ijcai]) *Sorts*
Facts	

Screen 2

After defining another predicate ("small"), the has now used the previously defined predicate "large" in an input fact. Note again that the sorts window is immediately updated to show the membership of the term "ijcai" in the sort "event". A predicate is not required to be defined before it is used - the user is free to ignore this integrity constraint (Screen 3):

BLIP TU	
large(ijcai) conference(ijcai) <=	large(<event>) small(<event>) conference(<event>) *Predicates*
Facts	subsort(event,all) sort(event,[ijcai]) *Sorts*

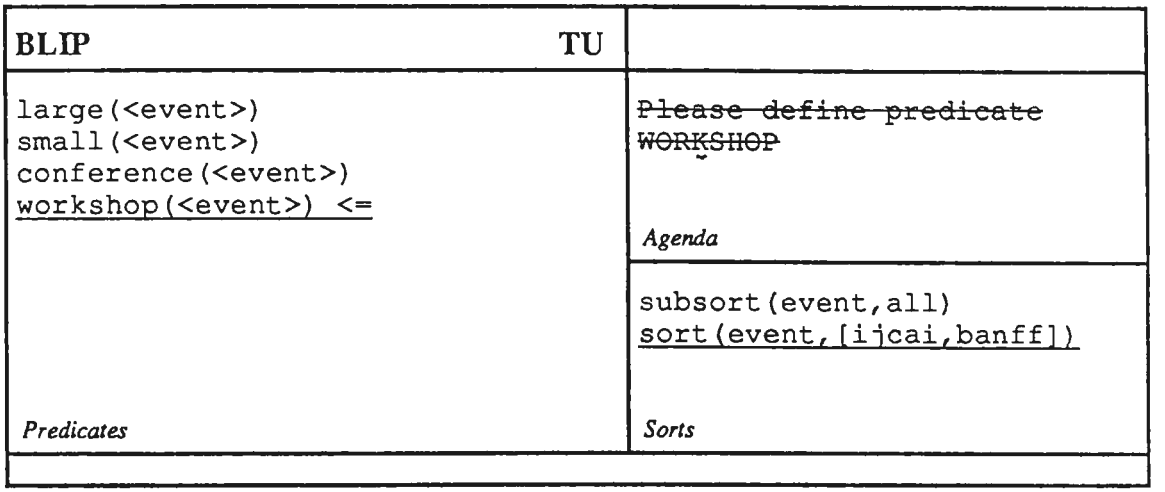
Screen 3

Here, the new predicate "conference" was used with an argument the sort assigment of which is already known ("ijcai"). Based on this information, the system has exploited the integrity constraint that each term be a member of only one sort, and has generated the predicate definition for "conference" automatically.

BLIP TU	
large(ijcai) conference(ijcai) workshop(banff) <=	Please define predicate WORKSHOP *Agenda*
Facts	subsort(event,all) sort(event,[ijcai]) *Sorts*

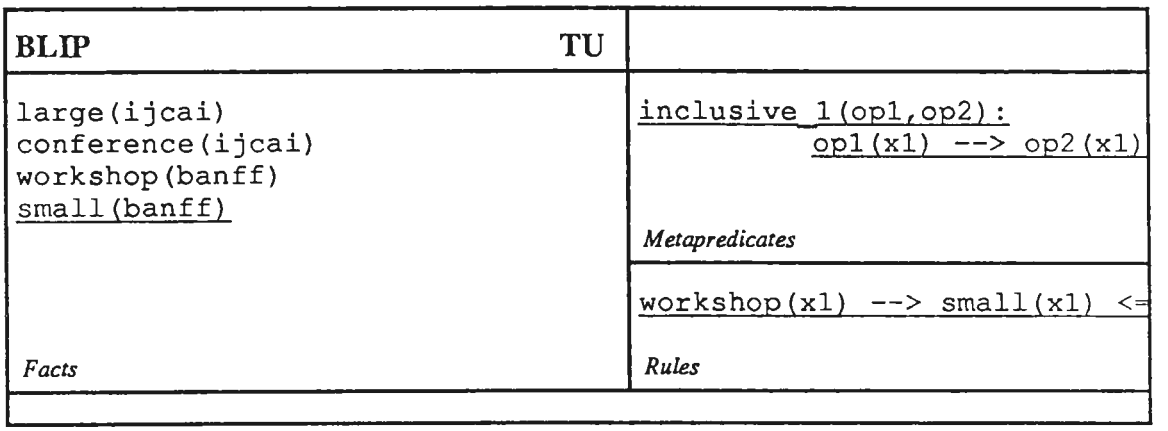
Screen 4

The user has now input a fact where neither the predicate, nor the argument is known, so the integrity violation cannot be resolved. Instead, the violation has been recorded on the integrity maintenance agenda. Note that even though the new fact has been recorded in the facts window, the sort assignment for "banff" could not be performed yet.



Screen 5

The user has now taken care of the predicate definition, so the previously unresolved sort assignments have been completed, and the agenda entry has been removed automatically. The user's attention now turns to *rules* (Screen 6):



Screen 6

Recall that in BLIP, each rule is also expressed by a corresponding metafact. Since we started with an empty knowledge base, BLIP could not find an appropriate metapredicate, and the following dialogue took place in the rule window:

```
     No matching metapredicate found.
     Abstracted metapredicate:
     m1(op1,op2): op1(x1) --> op2(x1)
     Entering metapredicate. Choose a different name?
> inclusive_1
     Trying to integrate new metapredicate into
     metametaknowledge ... No metametafacts generated.
```

The system has automatically generated a new metapredicate and let the user choose the name. The following attempt to derive metametafacts that might characterize the new metapredicate fails, since no metametapredicates are known. After entering the new metapredicate, BLIP reexpresses the input rule as a metafact, and immediately uses the

rule for inferring a new fact, resulting in screen 6. (The new metafact is displayed on screen 7).

If desired, the user can also use BLIP's metaknowledge directly to input rules:

BLIP TU	
large(ijcai) conference(ijcai) workshop(banff) small(banff) crowded(ijcai) *Facts*	inclusive_1(workshop,small) inclusive_1 (conference, crowded) <= *Metafacts* workshop(x1) --> small(x1) conference(x1)--> crowded(x1) *Rules*

Screen 7

After inputting a metafact, the corresponding rule has been generated, and was used to derive a new fact.[6] If the user now decides to delete the newly entered metafact again, the recorded dependencies between items in different knowledge sources are used to make sure that all inferences based on this metafact are retracted:

BLIP TU	
large(ijcai) conference(ijcai) workshop(banff) small(banff) ~~crowded(ijcai)~~ *Facts*	inclusive_1(workshop,small) ~~inclusive_1~~ ~~(conference,crowded)~~ *Metafacts* workshop(x1) --> small(x1) ~~conference(x1) --> crowded(x1)~~ *Rules*

Screen 8

6 Since the predicate "crowded" used in the conclusion is unknown, BLIP also generates an appropriate definition (not shown).

5. Discussion

The interaction shown above is of course only the beginning of an extended domain modeling process that we cannot follow here. Instead, let us turn to a more general discussion of BLIP's capabilities with respect to the design requirements for sloppy modeling systems that were developed in section 2.

Flexibility

Two factors contribute to the flexibility of the BLIP system. First, the interaction with the user is organized around independent commands that are available at any time, and not around extended structured dialogues. That lets the system easily follow sudden ideas on the part of the user. Second, and more importantly, the "agenda of open ends" allows the user to postpone treating integrity constraint violations and return to handle them later, thus making sure such tasks don't get in the way of domain modeling.

Reversability

BLIP is completely reversable with respect to the knowledge sources that are maintained by the inference engine, i.e., facts and rules at domain, meta, and metameta levels. For those knowledge sources, deletions and modifications are supported at any point. For other knowledge sources, those operations are supported, but not in the best possible way. Currently, modifications of predicates are supported only by first deleting the predicate and all facts using it, and then redefining it and reentering the facts. For obvious reasons, this is an unsatisfying way of modifying a definition, and a more automatic support would be desirable.

For metapredicates, the situation is even more difficult, because they are used in the learning algorithms. Fortunately, metapredicates are part of the domain-independent knowledge that is incorporated into the system, so that the need for modifications (apart from additions, which are possible at any point) does not arise in actual modeling situations.

Another weak point of the current system is the lack of support for large-scale revisions to the structure of the model, such as changing the arity of a predicate. If new ideas about the structure of a model are to be incorporated even at later stages, such facilities must be available. One possibility of supporting large-scale changes would be an extended macro-facility, such as the one built into the KREME system [Abrett and Burstein 86].

Integrity and Consistency Maintenance

Consistency maintenance is guaranteed in BLIP by the reason maintenance procedures of its inference engine. Through the procedures in the coordinator module, integrity maintenance between knowledge sources is also fully supported. The "agenda of open ends" provides a convenient way for the user to take care of integrity problems.

As an interesting topic for future research, we should point out that up to now, we have not achieved a declarative representation of the dependencies between knowledge sources. Such a representation would allow a generic integrity maintenance tool to be built, and new dependencies could be added simply by modifying the integrity constraint data base. Currently, this requires modifying several procedures in the coordinator.

Liveness

All displays used in the BLIP interface are completely live; all changes to the model being built are immediately processed by inference engine and learning algorithms and reflected in the display. A slightly confusing feature for new users is the forward-chaining depth limit, which causes some expected inferences not to appear in the display. We are thinking about a more flexible limit which will take into account the kind of rule used for inferences (e.g., not counting metarule applications). The graphical displays need to be explicitly updated right now, but we intend to fix that in the near future.

Inspectability

The text and graphics based display windows provide a good way of looking at the contents and structure of a domain model. The focus facility is a first step towards managing large domains. One major weakness, however, is that it does not exploit the dependencies between knowledge sources to automatically focus other windows on the appropriate items when a focus is selected for one knowledge source. That task is left to the user. Also, it would be desirable to offer other criteria for focusing than the syntactical ones available now. Interesting possibilities include focusing on certain inference chains, on unused predicates, or on predicates for which no rules could be discovered by the system. Such facilities would be one further step in the direction of *knowledge analysis*, with automatic facilities for analyzing the structure of a domain model and discovering imbalances and gaps.

Transitionality

The BLIP system is transitional both in the traditional sense and in the sloppy-modelling sense. First, it allows users to begin working in a domain-oriented style by just inputting facts, predicates, and rules. This is supported by providing tools that automatically reexpress rules in the metafact-representation needed by the system. Second, and perhaps more importantly, in BLIP, the same representations and knowledge sources are manipulated by the user and the learning algorithms that support the system's part of the sloppy modeling process. That way, the interaction with the system is always the same, whether the automatic modeling tools are used or not.

6. Conclusion

Sloppy modeling is a paradigm for machine-supported knowledge acquisition that explicitly denies using structured elicitation dialogues for guiding the user through the knowledge acquisition process. Instead, by constructing an extremely flexible system, the user is provided a possibility to perform an unconstrained, exploratory first modeling phase with the help of the system. In this paper, we have shown that assuming such a knowledge acquisition paradigm leads to several tough requirements that a knowledge acquisition environment must meet: it must be flexible, because the user does not enter his or her knowledge in a well-organized fashion; instead, sudden ideas and jumps are common. It must be reversable, because the user will change his or her mind. It must support integrity and consistency maintenance and not burden the user with those chores. It must be live, because the user needs immediate feedback on the consequences of modelling decision. It must guarantee the inspectability of the emerging model, and

finally, be transitional. The BLIP system, which we then presented and evaluated on the above criteria, is a significant first step towards such a sloppy modeling environment, even though there is still a long way to go, especially in the area of knowledge analysis.

Nevertheless, the work reported here points the way towards a new style of interacting with a knowledge acquisition system. The system is no longer a mere tool that the user has to control and use, and the user is no longer just regarded as an oracle that answers the system's questions. Instead, both act as partners that cooperate and interact in the common problem solving activity of building up a well-structured model of the domain.

Implementation Note

BLIP is implemented in Prolog and Lisp on a Symbolics 3670.

Acknowledgements

I am grateful to the other members of the KIT-LERNER project, Werner Emde, Katharina Morik, and Sabine Thieme for their ideas, comments, and criticism. Many of the ideas reported in this paper are theirs. The interface described here was designed and implemented by Sabine Thieme and the author.

References

Abrett, Glenn, and Burstein, Mark H., "The KREME Knowledge Editing Environment," in *Proc. Knowledge Acquisition for Knowledge-Based Systems Workshop*, pp. 1-0 - 1-19, Banff, Canada, November, 1986. (Also to appear in *Int. Journal of Man Machine Studies*)

Badre, Albert N., "Designing transitionality into the user-computer interface," in *Human-computer Interaction*, edited by G. Salvendy, Elsevier Publishing, 1984.

Boose, John H., and Bradshaw, Jeffrey M., "Expertise Transfer and Complex Problems: Using AQUINAS as a Knowledge Acquisition Workbench for Expert Systems," in *Proc. Knowledge Acquisition for Knowledge-Based Systems Workshop*, pp. 5-0 - 5-16, Banff, Canada, November, 1986. (Also to appear in *Int. Journal of Man Machine Studies*)

J., William, "From GUIDON to NEOMYCIN and HERACLES in Twenty Short Lessons: ONR Final Report 1979-1985," *The AI Magazine*, pp. 40 - 60, August, 1986.

Eshelman, Larry, and McDermott, John, "MOLE: A Knowledge Acquisition Tool that Uses its Head," in *Proc. AAAI-86*, Vol. 2, pp. 950 - 955, Philadelphia, PA, August, 1986.

Emde, Werner, "IM 2.0 - Eine Inferenzmaschine fuer ein inkrementell lernendes System," forthcoming.

Emde, Werner, and Morik, Katharina, "The BLIP System," Tech. Rep. KIT-REPORT 32, Techn. Univ. Berlin, Berlin, West Germany, February, 1987.

Langbein, Kurt, Martin, Hans-Peter, Sichrovsky, Peter, and Weiss, Hans, "Bittere Pillen," Kiepenheuer & Witsch, Cologne, FRG, 1983.

Michalski, R.S., and Chilauski, R.L., "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*, Vol. 4, No. 2, pp. 125 - 161, 1980.

Mitchell, Tom M., Mahadevan, Sridhar, and Steinberg, Louis I., "LEAP: A Learning Apprentice for VLSI Design," in *Proc. IJCAI-85*, pp. 573 - 580, Morgan Kaufman, Los Altos, CA, August, 1985.

Morik, Katharina, "Acquiring Domain Models," in *Proc. Knowledge Acquisition for Knowledge-Based Systems Workshop*, pp. 32-0 - 32-15, Banff, Canada, November, 1986. (Also to appear in *Int. Journal of Man Machine Studies*)

Morik, Katharina, "Anything you can do I can do meta," Tech. Rep. KIT-REPORT 40, Techn. Univ. Berlin, Berlin, West Germany, November, 1986.

Newell, Allen, "The Knowledge Level," *Artificial Intelligence*, Vol. 18, pp. 87 - 127, 1982.

Peltason, Christof, von Luck, Kai, Nebel, Bernhard, and Schmiedel, Albrecht, "The User's Guide to the BACK System," Tech. Rep. KIT-REPORT 42, Techn. Univ. Berlin, Berlin,, January, 1987.